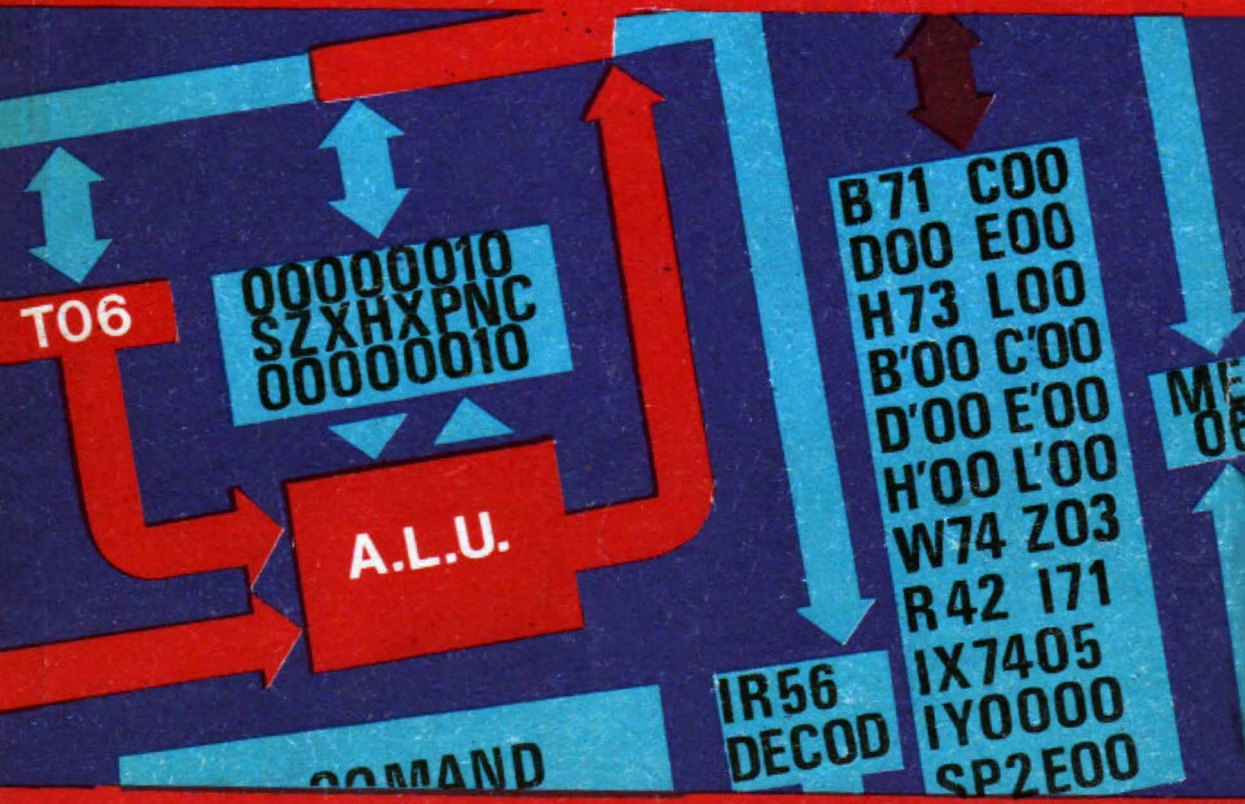
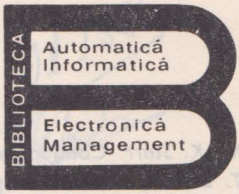
M. PATRUBÁNY

Volumul 1

TOTUL DESPRE... MICROPROCESORUL Z80



EDITURA TEHNICA



Ciclul „Totul despre ..."

Biblioteca de automatică, informatică, electronică, management. Seria PRACTICĂ

M. K. Starr. **Conducerea producției.**
T. Baron ș.a. **Calitatea produselor Manual practic.**
A. Vlădescu ș.a. **Radioreceptoare**
M. Mayer. **Tiristoare în practică. Mutatoare cu comutație forțată**
G. Moltgen. **Tiristoare în practică. Mutatoare cu comutație de la rețea**
L. Zamfirescu, I. Oprescu. **Automatizarea cuptoarelor industriale**
I. Papadache. **Automatica aplicată, ediția I și a II-a**
Șt. Alexandru. **Automatizarea proceselor tehnologice în industria lemnului**
V. H. Lisicikin. **Prognoza tehnico-științifică în ramurile industriei**
G. Raymond. **Tehnica televiziunii în culori**
T. Homoș. **Capacitatea de producție în construcții de mașini**
S. Radu, D. Filoti. **Centrale telefonice automate. Sisteme de comutație.**
R. Stere ș.a. **Tranzistoare cu efect de cîmp**
D. N. Sapiro. **Proiectarea radioreceptoarelor**
V. Antonescu, M. Popovici. **Ghid pentru controlul statistic al calității producției**
N. Stanciu ș.a. **Tehnica imaginii în cinematografie și televiziune**
P. Vezeanu, Șt. Pătrașcu. **Măsurarea temperaturii în tehnică**
T. Penescu, V. Petrescu. **Măsurarea presiunii în tehnică**
P. Popescu, P. Mihordea. **Măsurarea debitului în tehnică**
P. Vezeanu. **Măsurarea nivelului în tehnică**
C. Hidoș, P. Isac (coordonator,) **Studiul muncii, vol. I—VIII**
V. Baltac ș.a. **Calculatorul FELIX C-256, Structură și programare**
R. L. Morris. **Proiectarea cu circuite integrate TTL**
Ishikawa Kaoru. **Controlul de calitate pentru maiștri și șefi de echipe**
A. M. Buhtiarov ș.a. **Culegere de probleme de programare**
P. Constantinescu. **Sisteme informatice, modele ale conducerii și sistemelor conduse**
E. S. Buffa. **Conducerea modernă a producției, vol. I și II**
A. Vătășescu ș.a. **Dispozitive semiconductoare. Manual de utilizare**
A. Nadolo. **Măsurarea volumului și cantității lichidelor în industrie**
Ch. Jones. **Design. Metode și aplicații**
Gh. Pisău ș.a. **Elaborarea și introducerea sistemelor informatice**
C. Hidoș. **Analiza și proiectarea circuitelor informaționale în unitățile economice**
A. Vătășescu ș.a. **Circuite integrate liniare. Manual de utilizare vol. 1, 2, 3, 4**
M. Silișteanu ș.a. **Scheme de televizoare, magnetofoane, vol. 1 ș, 2 ed. a II-a**
D. W. Davies. **Rețele de interconectarea calculatoarelor**
V. Pescaru ș.a. **Fișiere, baze și bănci de date**
Gh. Baștiurea ș.a. **Comanda numerică a mașinilor-unelte**
N. Sprînceană ș.a. **Automatizări discrete în industrie. Culegere de probleme**
M. Florescu. ș.a. **Cibernetică, automatică, informatică în industria chimică**
S. Călin. **Optimizări în automatizări industriale**
S. Maican. **Sisteme numerice cu circuite integrate**
I. Ristea ș.a. **Manualul muncitorului electronist**
M. Simionescu. **Proiectare unitară a circuitelor electronice**
C. Cluceru. **Tehnica măsurărilor în telecomunicații**
P. Nițulescu. **Electroalimentarea instalațiilor de telecomunicații**
R. Rapeanu ș.a. **Circuite integrate analogice. Catalog**
Șt. Lozneanu ș.a. **Casetofoane. Depanare. Funcționare**
T. Rădulescu ș.a. **Centrale telefonice automate**
N. Iosif ș.a. **Tiristoare și modele de putere. Catalog**
P. Postelnicu. **Sisteme și linii de transmisiuni telefonice**
M. Silișteanu ș.a. **Receptoare TV în culori**
V. Baltac ș.a. **Sisteme interactive și limbaje conversaționale**
V. Baltac ș.a. **Calculatoare electronice, grafica interactivă, prelucrarea imaginilor**
T. Geber, I. Miu ș.a. **Echipamente periferice 1, 2, 3**
A. Davidoviciu, B. Bărbat. **Limbaje de programare pentru sisteme în timp real**
M. Guran, Fl. Filip **Sisteme ierarhizate în timp real cu prelucrarea distribuită a datelor**
În ciclul Totul despre... a apărut: A. Petrescu ș.a., Totul despre... calculatorul personal a MIC

Patrubány Miklós

TOTUL DESPRE... MICROPROCESORUL Z80

Volumul 1

EDITURA TEHNICĂ
București, 1989

Prefață: ing. GH. CONSTANTINESCU
Recenzii: dr. ing. ADRIAN DAVIDOVICIU,
dr. ing. NICOLAE ȚĂPUȘ
dr. ing. LIVIU DUMITRAȘCU,

Redactor: ing. PAUL ZAMFIRESCU

Toate drepturile pentru această ediție
(cărți + casetă cu produsele-program
BAIEM PP01 și BAIEM PP02)
rezervate Editurii Tehnice, Piața
Scînteii 1, București, Romania.

Coperți: arh. SILVIA PINTEA,
arh. LIVIU DERVEȘTEANU (ilustrații)
Tehnoredactor: V. E. UNGUREANU
C.Z.: 681.3
Bun de tipar: 6 05 1989
Coli de tipar: 28

ISBN 973-31-0008-0
ISBN 973-31-0007-2

ÎNTREPRINDEREA POLIGRAFICĂ CLUJ,
Municipiul Cluj-Napoca, cd. nr. 425/1987

PREFAȚĂ

Microprocesorul, creație ce a răsărit în constelația siliciului începînd cu deceniul al 8-lea al secolului XX, a cunoscut o evoluție spectaculoasă, atingînd în prezent cotele maturității, în ciuda tinereții sale. Procesul evolutiv se datorește, în principal, rafinamentelor introduse în tehnologia prelucrării siliciului, nefiind — însă — de neglijat aportul noilor arhitecturi logice a procesoarelor și reacția utilizatorilor acestora.

În cursa microprocesoarelor s-au angajat mai mulți producători de componente electronice, avînd loc o emulație și o selecție a soluțiilor tehnice adoptate. Pentru clasa microprocesoarelor de 8 biți cursa a fost cîștigată de microprocesorul Z80 lansat de firma ZILOG, denumit MMN 80 CPU de către întreprinderea românească Microelectronica. Prezent în milioane și milioane de aplicații pe toate meridianele globului acest microprocesor dispune de o vastă acoperire cu programe de uz general sau de aplicație și de o impresionantă bibliografie.

În țara noastră, grație dezvoltării impetuoase a industriei electronice în ultimele două decenii, microprocesorul Z80 este puternic implantat, constituind elementul esențial al multor echipamente de tehnică de calcul și automatizare cum ar fi calculatoarele personale PRAE, aMIC, HC-85, TIM-S, microcalculatoarele ECAROM, MADS, MIND, videoterminalele DAF 2020, VDT 52S, etc. Microprocesorul MMN 80 CPU și circuitele MOS-LSI din familia sa MMN 80 SIO, MMN 80 PIO, MMN 80 CTC și MMN 80 DMA se află în producție de serie la întreprinderea Microelectronica. Aceste realizări ale tinerei industrii electronice românești se găsesc în exploatare în mai toate domeniile vieții economice și sociale, formînd obiectul atenției zilnice a mii și mii de specialiști.

În acest context, prezenta lucrare constituie un prețios material de instruire și perfecționare pus la dispoziția utilizatorilor și constructorilor echipamentelor ce utilizează microprocesorul MMN 80 CPU.

Nota de originalitate deosebită a cărții constă în prezentarea modului de funcționare a microprocesorului cu ajutorul unui program de simulare, **care permite vizualizarea modificărilor stărilor interne pe măsura execuției ciclurilor funcționale. În acest scop, cartea este însoțită (o parte a tirajului) de un suport magnetic conținînd programul de simulare, fiind astfel o apariție inedită în peisajul editorial.**

Cele două volume conțin un bogat bagaj de cunoștințe, plăcut și riguros prezentate, toate utile celor ce vor să se inițieze în tainele lui Z80, precum și celor deja inițiați în acest domeniu.

Putem afirma că această carte conține totul (și chiar mai mult) despre Z80, avînd în vedere, de exemplu, setul de instrucțiuni nedeclarate de firmă, dar descoperite în țară și prezentate în capitolul G al părții a II-a.

Exersarea **exemplului de proiectare** hardware-software al unui echipament cu Z80, căruia îi este consacrat volumul 2, oferă cititorului un îndrumar practic bine conturat pentru abordarea coerentă a unor proiecte bazate pe principiul logicii microprogramate.

Recomandăm cu căldură această lucrare tuturor celor ce utilizează sau vor utiliza microprocesorul Z80 sau alți confrați ai săi. Sîntem convinși că, prin maniera tratării subiectului, cartea de față va contribui substanțial la răspîndirea și asimilarea rapidă a cunoștințelor din acest domeniu de maxim interes pentru dezvoltarea economică și socială a patriei.

Adresăm pe această cale felicitări redacției de informatică și tehnică de calcul a Editurii Tehnice, pentru inițiativa de a publica cărți de profil însoțite de suport magnetic, cum și autorului pentru frumoasa și utila carte și urăm cititorului o lectură plăcută și instructivă.

București, 3 mai, 1989

ing. GH. CONSTANTINESCU
Directorul Intreprinderii
Microelectronica

CUVÎNT ÎNAINTE

Gînduri la editarea de către Editura Tehnică a primei cărți românești însoțite de o casetă cu programe de calculator

Cîte cineva și-ar putea pune întrebarea dacă în anul 1989 se mai justifică editarea unei cărți dedicate unui microprocesor de 8 bit, microprocesorul Z80.

Acest dubiu pare a fi susținut de numărul relativ mare de cărți care tratează această clasă de microprocesoare, apărute mai ales în străinătate, dar și în țară, cum și de faptul — că între timp — microprocesoarele de 16 și 32 bit cîștigă din ce în ce mai mult teren.

Microprocesorul Z80 nu este, însă, un oarecare. In afară de faptul că — pe plan mondial — a echipat numeroase calculatoare personale și o multitudine de echipamente, în țara noastră este creierul „electronic" al calculatoarelor personale românești PRAE, aMIC, HC—85, TIM—S ș.a., este fabricat pe scară largă de Uzina Microelectronica și constituie un component esențial al multor echipamente românești electronice și de automatizare.

Elementul care ne-a dat o mai mare siguranță în abordarea acestei lucrări l-a reprezentat existența unor variante ale actualului program Visible—Z80, concepute de autor, în cadrul colectivului de la ITCI—Cluj-Napoca.

Am considerat acest program — elaborat iterativ și finisat printr-o muncă de șapte ani — un mijloc care permite cunoașterea completă a funcționării microprocesorului Z80. Iar, de aici, înțelegerea altor microprocesoare va fi relativ simplă. Astfel, un alt microprocesor, mai simplu dar de mare răspîndire în lume și în România, este I8080 ; cunoașterea „fratelui mai mare Z80" constituie — implicit — o deplină stăpînire a lui I8080, cu care compatibilitatea unilaterală este rezolvată. Dar nu numai înțelegerea microprocesoarelor mai simple ci și a celor mai complicate, cu care se dotează (dar, mai ales, se prevede dotarea) noile calculatoare personale românești, devine perfect posibilă.

Visible—Z80 desenează pe ecranul televizorului sau monitorului unui calculator personal PRAE sau aMIC structura internă a microprocesorului Z80, ilustrînd prin grafică dinamică și semnale sonore toate activitățile desfășurate de acesta, atunci cînd el execută oricare din cele aproximativ 700 de instrucțiuni pe care le recunoaște.

Prin modul de abordare și prin facilitățile pe care le oferă, acest program poate fi considerat un unicat în materie, gîndit și realizat pentru o mai bună instruire (asistată de calculator) la nivel liceal și universitar, pentru oricine care utilizează sau — în mod necesar — va utiliza microprocesoarele.

Inițial, variante preliminare ale programului Visible—Z80 au fost implementate pe calculatorul personal PRAE—M, fiind preconizate a deveni „port fanion"-ul softwareului utilizator al acestui calculator, realizat în cadrul aceluiași colectiv. Am testat utilitatea programului în procesul de instruire, pe grupe de elevi și studenți, folosind primele versiuni operaționale ale programului. Considerînd rezultatele obținute drept bune, ne-am propus răspîndirea pe scară largă a programului, dorind să venim pe această cale în ajutorul tuturor celor care doresc să învețe utilizarea microprocesorului Z80. In aceeași idee, am reconceput programul și pentru un alt calculator personal din România, aMIC. Nu am reușit aceeași operație pe calculatoarele

compatibile cu calculatorul Sinclair ZX Spectrum (ex. HC—85, TIM—S), datorită imposibilității de a încadra imaginea generată de Visible—Z80 în ecranul mai restrîns al acestor calculatoare (192×256 puncte, față de 256×256 la PRAE și aMIC).

*

Pe acest drum ne-am întîlnit cu ideile redacției de informatică și tehnică de de calcul a Editurii Tehnice, care urmărea de mai mult timp să editeze primele cărți din România însoțite de casete magnetice capabile să acționeze calculatoare personale. Redactorul de specialitate a afirmat că, în 1980—81, cînd, aflat în Franța, elabora pentru concernul de calculatoare IBM—France și pentru editorul internațional Hachette o analiză de sistem în domeniul editării-fabricării-difuzării asistate a cărții, a devenit preocupat și de frămîntările din „interiorul" acestui uriaș editor (atunci, al 4-lea din lume) privind lansarea primelor cărți cu casete de calculator asociate; deși o reclamă pentru "... o ediție analizată pe calculator a lui Don Juan ..." trona de mult în vitrinele de pe Bd. Saint—Germain din Paris, vînzarea nu o lua din loc — spre bucuria concurenței americane, aflate în avans, ca și a celei franceze — datorită unor probleme ca: prețul, protecția la copiere a casetei, drepturile de autor, de copyright, de licențe ș.a., fapt care trăda că „drumul nou și pentr-un uriaș e greu".

Asemenea preocupări s-au putut concretiza la noi odată cu existența unor serii ale primelor calculatoare personale românești, PRAE și aMIC.

In acest climat favorabil, după consultări prealabile, am depus editurii, în primăvara lui 1987, manuscrisul lucrării axat pe un volum, constînd din două părți și ample anexe.

Mai apoi, în toamna aceluiași an, după o conlucrare adîncită cu redacția, intra în tipografie manuscrisul unei cărți în două volume și trei părți, din ciclul „Totul despre . . .", cu o structurare mai bună și mai „prietenoasă" față de un cerc larg de cititori, cu un ultim capitol complet nou și cu evidențierea prin multiple mijloace (inclusiv culoarea) a unor accente semnificative ale unei ediții unicat; între timp am colaborat cu Electrecord-ul pentru punerea la punct a electronicii înregistrărilor pe casete audio dedicate calculatoarelor. In primăvara lui 1988 editura a obținut pentru carte și casete tiraje apreciabile, iar întreg textul cules a fost corectat; pregătirea pentru tiparul ofset în culori, găsirea unor materiale adecvate, realizarea casetelor dar, mai ales, „frămîntări economice editoriale" (asemănătoare, oare, celor de la Hachette?) ne-au adus în acest an.

Cînd ... revenim puțin la „disecarea" complexei cărți aflate acum în fața cititorilor.

*

Inițial m-am gîndit la un manual de utilizare, bine exemplificat, al casetei VISIBLE—Z80. Am renunțat relativ repede la această idee, considerînd că numărul celor care doresc să-și aprofundeze cunoștințele despre microprocesorul Z80 este mult mai mare decît al celor care acced la calculatoarele personale PRAE și aMIC. Trebuia, deci, scrisă o lucrare care să le permită și celor care nu acced la calculatoare personale condiții bune de autoinstruire. Astfel am decis — așa cum a fost și opțiunea redacției — să introducem în text un număr mare de imagini, reprezentînd secvențe similare cu cele generate de Visible—Z80 pe ecranul atașat unui calculator personal, astfel încît caseta să poată fi înțeleasă și în lipsa calculatorului (deci, numai din textul, figurile și programele imprimate).

Am recurs la un stil, pe alocuri, eseist — mai rar întîlnit în literatura tehnică — dorind pe această cale să „colorez" uniformitatea aparentă a unui volum mare de imagini, în care doar cititorul atent putea sesiza diferențele. Am încercat să infiltrez picătură cu picătură, — fără a conturba continuitatea textului de bază — experiența acumulată în 10 ani de proiectare în domeniul microprocesoarelor. Am rezolvat acest lucru, de exemplu, înserînd informațiile dorite sub forma unor remarci, pe care le veți regăsi numerotate de la 1 la 67.

Din dorința de a veni în ajutorul cît mai multor cititori posibili (chiar și profesioniști) am elaborat încă o parte — inclusă ca parte a doua în volumul I sub titlul Complemente *—, din care aș dori să remarc studiul comparativ al semanticii limbajelor de asamblare 8080 și Z80, menit să fundamenteze mai bine noțiunea de limbaj de asamblare, cum și listele instrucțiunilor microprocesorului Z80 (atît a celor declarate cît și a celor ascunse), grupate în ordine alfabetică și în ordinea crescătoare a codurilor. Aceste din urmă liste au reprezentat un adevărat calvar. Ele au trecut prin 16 corecturi, la fiecare trecere găsindu-se încă erori. Nutrim speranța de a le fi eliminat pe toate, dar „tăria" afirmației o simțim puternic afectată, văzînd prin cîte am trecut!*

Dar, rolul esențial al complementelor pentru cititorii care au acces la un calculator personal PRAE sau aMIC este manualul de utilizare al casetei VISIBLE—Z80, care prezintă sistematic — pas cu pas — ce trebuie făcut pentru lucrul cu acest pachet de programe de simulare.

Recitind materialul rezultat, mi-am spus: „Toate bune". Cititorul poate afla totul despre modul în care microprocesorul Z80 execută instrucțiunile. Dar cum va ști el să-l utilizeze, să-l programeze? Răspunsul la această întrebare s-a putut da doar elaborînd un al doilea volum.

În ceea ce privește conținutul acestui volum, am hotărît să nu-l constitui dintr-un colaj de programe, adunate astfel încît să acopere cît mai bine setul de instrucțiuni al microprocesorului. Am analizat îndelung problema, știind că programele pe care le voi elabora vor trebui să se constituie într-un ansamblu unitar, care să aibă o finalitate. Inițial m-am gîndit la elaborarea unui program monitor (se regăsește pe aproape toate microcalculatoarele) — dar am renunțat la idee, datorită spectrului tematic relativ restrîns care putea fi acoperit (lipsa aproape totală a problemelor de interfață — hardware comandat prin program), și datorită faptului că nu aș fi putut spune mai nimic nou.

M-am jucat apoi cu ideea realizării software-ului pentru o imprimantă matricială. Spectrul tematic al acesteia este mai bogat, dar nici aici nu apar calcule matematice. Ori, această clasă de probleme este una din cele care se pot rezolva destul de elegant cu microprocesoare.

Continuîndu-mi căutările am ajuns la casa de marcat electronică. M-am bucurat mult văzînd că ea poate îngloba întregul proiect de imprimantă, suplimentîndu-l cu probleme de aritmetică și numeroase considerente de ergonomie.

S-a născut astfel volumul al doilea, axat în special pe programarea microprocesorului Z80 (software), în care prezint toate etapele de proiectare a unei case elec-

tronice de marcat virtuale, începînd cu specificația ei tehnică și terminînd cu documentația completă a programului.

Am început volumul cu un capitol introductiv, prezentînd noțiuni de programare structurată și făcînd recomandări de ordin general (uneori chiar și detalii de tehnică) privind modalitățile de realizare a unor programe cu „viață lungă". Pe parcursul elaborării proiectului am încercat să respect întocmai recomandările formulate în capitolul introductiv. Trebuie să recunosc că acest lucru nu a fost deloc ușor.

Stilul folosit în acest volum este mai riguros, el solicitîndu-l probabil mai mult pe cititor. Am încercat să înlesnesc asimilarea informațiilor, redînd majoritatea programelor elaborate sub trei forme : organigramă, limbaj de nivel înalt (pseudo—Pascal) și limbaj de asamblare.

Predau cartea cititorilor, puțin emoționat, adresînd-o atît celor începători cît și celor avansați !

Mulțumiri

Doresc să-mi exprim profunda gratitudine prietenilor mei, care m-au ajutat atît de mult în realizarea manuscrisului și produselor-program.

Inainte de toate țin să mulțumesc ing. Schneider Hans Rudolf, fără a cărui muncă plină de devotament programul Visible—Z80 n-ar fi fost ceea ce este.

Aceleași sentimente de recunoștință le port ing. Hidán Levente pentru ajutorul acordat la elaborarea studiului de caz din volumul 2.

Șirul celor ce m-au ajutat — cu sugestii, sfaturi și participare, — la punerea la punct a programelor și a manuscrisului, a filelor și listelor de instrucțiuni, a figurilor și imaginilor, a casetei este lung. Chiar și cu riscul de a omite pe unii, țin să le mulțumesc nominal: ing. Dorin Barbul (CCMIM Baia Mare), mat. Kiss Alexandru, ing. Pop Baldi Nicolae, ing. Teodora Rațiu, ing. Imecs Márton (Muzeul de istorie Cluj-Napoca), ing. Mirela Arion, ing. Radu Arsinte, fiz. Szász Detre, ing. Dinu Budura (Inst. Politehnic Timișoara), ing. Papp Hajnalka (IEIA Cluj-Napoca), ing. Ferencz Attila, ing. Sólyom Péter (Intr. Republica Reghin), ing. Elena Ruginã (Intr. Electrecord), cu o remarcă aparte privind adaptarea instalației de înregistrare pentru caseta de calculator. Cei mai mulți, colegi la filiala ICSIT-TCI

Îmi exprim aprecierea pentru spiritul creator, atmosfera fertilă instaurată de către conducerea Institutului de Cercetare Științifică și Inginerie Tehnologică pentru Tehnică de Calcul și Informatică (ICSIT—TCI) București, — ing. Victor Megheșan, dr. ing. Dan Roman, dr. Emil Munteanu (conducătorul filialei din Cluj-Napoca) — mediu de lucru care oferă spațiu larg de desfășurare profesională tuturor tinerilor specialiști.

In sfîrșit, dar nu în cele din urmă, trebuie să mulțumesc Editurii Tehnice pentru inițiativa de a lansa un asemenea gen de ediție, în premieră românească.

Cluj-Napoca, 6 mai 1989 AUTORUL

DESPRE CARTE ŞI DESPRE AUTOR

Editura tehnică dorea de mult timp (chiar înaintea apariţiei primelor calculatoare personale româneşti, PRAE şi aMIC) să editeze, în redacţia de informatică şi tehnică de calcul, primele cărţi cu produse-program asociate, pe mediile magnetice disponibile (casete audio, dischete).

Prima ofertă acceptată a constituit chiar de la început o reală surpriză, calitatea produsului-program oferit — un simulator audio-vizual complex al funcţionării microprocesorului Z80 şi un proiect complet al unei case de marcat electronice — depăşind cu mult aşteptările de programe originale pentru calculatoare personale.

Este adevărat că pachetul de programe — denumit VISIBLE-Z80 — a fost conceput chiar de către autorul calculatorului personal românesc PRAE, care l-a transpus ulterior şi pe celălalt calculator personal românesc aMIC, primele produse de acest fel din ţară.

Cartea care s-a dezvoltat în jurul acestui nucleu esenţial a fost concepută astfel încît funcţionarea microprocesorului Z80 şi proiectarea echipamentelor cu microprocesoare Z80 să fie înţelese indiferent dacă cititorii au sau nu acces la calculatoarele personale amintite.

În esenţă, **Visible-Z80** desenează pe ecranul unui televizor ataşat la un calculator personal structura internă a microprocesorului Z80 şi simulează prin mişcare şi sunet toate activităţile care au loc în interiorul microprocesorului, atunci cînd el execută oricare din cele 696 de instrucţiuni publicate, pe care microprocesorul Z80 real le poate recunoaşte şi executa.

În carte, setul de instrucţiuni şi hardware-ul microprocesorului sînt prezentate cu 225 secvenţe grafice indentice cu cele generate de simulatorul Z80,

Activităţile majore din interiorul microprocesorului sînt subliniate prin efecte sonore specifice, permiţînd utilizatorului identificarea lor simplă. Simularea grafică a tuturor instrucţiunilor poate fi executată în 7 regimuri de simulare distincte, care permit adaptarea vitezei de simulare la cunoştinţele fiecărui utilizator în parte, ea fiind foarte gustată de cei care doresc să se autoinstruiască.

Existenţa regimului de simulare Automat (simularea decurge automat cu o viteză programabilă) şi a **celui Manual** (simulatorul se opreşte la sfîrşitul fiecărui ciclu, aşteptînd apăsarea unei taste a calculatorului) facilitează utilizarea lui Visible-Z80 de către profesori, ei putînd da — în pauzele indefinit de lungi ale regimului manual — toate eventualele explicaţii necesare unei mai bune înţelegeri a fenomenelor demonstrate.

Visible-Z80 poate fi folosit ca instrument de lucru şi de utilizatorii avansaţi, conţinînd, pe lîngă simulatorul grafic propriu-zis şi supervizorul de simulare, încă trei componente software majore : monitorul ,asamblorul şi dezasamblorul.

In prima parte a **volumului 1** se prezintă exhaustiv structura şi funcţionarea amănunţită a microprocesorului, iar în a doua parte, denumită „Complemente . . . " sînt grupate manualul de utilizare a casetei magnetice, comparaţia între limbajele de asamblare a două microprocesoare uzuale (Z80 şi i8080), fişele detaliate ale tuturor instrucţiunilor microprocesorului Z80, pe clase şi grupe, lista instrucţiunilor publicate de firmă si a celor descoperite de utilizatori (inclusiv în colectivul ITCI Cluj-Napoca unde activează autorul).

Volumul 2 este un studiu de caz foarte dezvoltat, de proiectare a unei case de marcat electronice, cu microprocesor Z80, încheiat cu proiectul complet al casei de marcat scris în limbaj de asamblare, comentat, aproape excesiv, nu numai pe module ci chiar pe fiecare instrucţiune. Acest proiect este încărcat în cod binar şi pe caseta magnetică. Se realizează astfel un excelent manual de instruire şi de lucru pentru proiectanţii de aplicaţii, care sînt puşi la încercare în ultimul capitol al cărţii de o temă, la a cărei rezolvare, ecranul televizorului ataşat calculatoarelor personale amintite devine „casa de marcat" proiectată.

Ing. PATRUBÁNY MIKLÓS, este cercetător ştiintific la Filiala din Cluj-Napoca a Institutului de Tehnică de Calcul şi Informatică (ICSIT—TCI) şi cadru didactic asociat la Institutul Politehnic din Cluj-Napoca. Este autorul unuia dintre primele două calculatoare personale româneşti, şi anume a calculatorului PRAE-M, pe care l-a prezentat specialiştilor la „Cercul utilizatorilor de Microcalculatoare şi Terminale Programabile", în noiembrie 1983, la Buşteni. După ce a absolvit Liceul Stephan Ludwig Roth (Mediaş, 1971), cîştigă prima ediţie a Olimpiadei profesionale studenţeşti „Traian Lalescu" (Timişoara, 1973)", ca student al Facultăţii de Electrotehnică din Cluj-Napoca. După un stagiu la secţia de circuite integrate a Intreprinderii de Piese Radio şi Semiconductoare — Băneasa, îşi continuă activitatea profesională la actualul loc de muncă, conducînd elaborarea unei familii de calculatoare personale (Prae—M, Prae—Max, Prae—Phoenix) şi produse software (MATH—I, MATH—II), publicînd totodată numeroase lucrări ştiinţifice în ţară şi în străinătate.

Cuprins general

Volumul 1

Prefață V
Cuvînt înainte VI
Despre carte și despre autor X
Cuprins general XI
Despre carte în limba engleză. Cuprins general în engleză XIi
Cuprins detaliat vol. 1 XIX

Partea I

Microprocesorul Z80 : o prezentare inedită. Simularea funcționării lui Z80, cu și fără casetă magnetică. Aventuri cu Visible—Z80 . . . XXIII

0. Praeludium. Calculatoare (retrospectivă) și microprocesoare. . . . 1
1. Structura internă. 12
2. Semnalele externe 31
3. Setul de instrucțiuni 38
4. Cicluri de bază 57
5. Instrucțiuni de transfer 73
6. Instrucțiuni aritmetice și logice . 120
7. Instrucțiuni diverse 144
8. Instrucțiuni de intrare/ieșire . . 176
9. Cicluri speciale 187

Partea a II-a

Complemente privind microprocesorul Z80 și simulatorul său 231

A. Manualul de utilizare a casetei de simulare Visible—Z80 . . . 233
B. Procedură de testare folosită pentru validarea programului Visible-Z 80 250
C. Comparație semantică a limbajelor de asamblare ale microprocesorului I8080 și Z80 257
D. Date de catalog ale microprocesorului Z80 263
E. Analiza setului de instrucțiuni ale microprocesului Z80 271
F. Lista celor 696 instrucțiuni declarate ale microprocesorului Z80 395
G. Cele 458 instrucțiuni ascunse ale microprocesorului Z80 420

Volumul 2

Cuprins general V
Cuprins detaliat vol. 2 VI

Partea a III-a

Programarea microprocesorului Z80. Un proiect : o casă de marcat electronică 1

10. Interregnum. Ghid de proiectare structurată 3
11. Specificația tehnică 17
12. Structura constructivă (Hardware) 25
13. Definirea structurilor de date . 60
14. Implementarea programului (Software) 78
15. Module software dedicate pentru testare 122
16. Inițializare și supervizare . . . 134
17. Lista comentată a programului . 141
18. Memento-ul proiectului 217
19. In loc de probleme și exerciții „Învierea casei" de marcat . . . 224

Bibliografie 232

Caseta cu produsele-program pentru calculatoarele personale PRAE și a MIC (pentru cărțile cu casetă)

M. PATRUBÁNY

ALMOST EVERYTHING ABOUT ... THE Z80 MICROPROCESSOR

TECHNICAL PUBLISHING HOUSE, BUKAREST, 1989

The publication of this book dedicated to the detailed presentation of a well proved microprocessor — Z80 — represents an editorial novelty in Romania. It is the first book published together with a program cassette.

The magnetic cassette contains two versions of Visible—Z80. This program, elaborated by a team led by the author, was implemented on two Romanian personal computers : Prae and aMIC. It is the result of about 10 man. years work. Visible—Z80 is a complex audio-visual simulator, whose succesive versions have been submitted to exhaustive tests. Now it represents an excellent means of computer aided instruction for all those who want to initiate themselves in the world of microprocessors.

Essentially Visible—Z80 draws on the screen of a TV-set connected to a personal computer, the internal structure of the Z80 microprocessor and it simulates through movement and sound all the activities that take place inside the microprocessor, when it executes any of the 696 published instructions which the Z80 microprocessor can really recognize and execute.

Main activities inside the microprocessor are underlined by specific sound effects which permit the user their simple identification. The graphic simulation of each instruction can be carried out in 7 distinct simulation modes. First of all there is the basic mode which is more detailed then the microprocessor's clock by clock cycle mode. There follow simulation modes going through the machine cycle mode and reaching two fast modes : the trace mode and the invisible mode. In each mentioned mode the simulation speed can be varied between high limits : 3 screen activities/s—0.3 screen activities/s. This facility permits the adaptation of the simulation speed to each user's knowledge level. This feature is adored especially by those who want to learn by themselves.

The existence of the automatic simulation mode and of the manual one enhances the use of Visible—Z80 by teachers. In the automatic mode simulation proceeds automatically with a programable speed, while in the manual one the simulator stops at the end of each cycle waiting until a key is pressed. In this latter mode — during the indefinitly long brakes —, teacher may give all the eventual explanations necessary for the better understanding of the demonstated phenomena.

Experimentation of Visible—Z80 with groups of pupils and students demonstrates a sensible diminution of the time necessary for study. In about two or three weeks of daily study Visible—Z80 users learn in detail the instruction set. This is an important step for all those who are going to write programs in assembly language. They can also accomodate with many of the Z80's hardware secrets, because Visible—Z80 is able to simulate the microprocessor's behaviour even in some specific hardware activities : RESET, BUSRQ, NMI, INT. The user can simulate the activation of any of these signals pressing a key (R, B, N or I), on the computer's keyboard, while Visible—Z80 executes the simulation of any instruction.

But Visible—Z80 isn't only an instrument for study. Beside the instruction simulator and the simulation supervisor, Visible—Z80 contains other three major software components :

- the enclosed monitor : is a useful program, that permits the direct acces to memory and to the microprocessor's internal registers. It is intended to aid debugging of programs written in assembly language ;
- the enclosed assembler : translates a program written in Z80 assembly language into machine code ;
- the enclosed disassembler : executes the inverse operation to the assembler.

This is why Visible—Z80 can be considered a working tool even by more advanced users.

The present book has been conceived in a way that it may allow to exploit the facilities offered by Visible—Z80 in the highest degree. Therefore, the reader may use a computer or not. It's so because the presentation of the instruction set and that of the microprocessor's hardware (volume 1) has been done with the help of 225 graphic sequences identical with those generated by Visible—Z80.

But knowledge of a processor's hardware and its instruction set isn't enough to be a good user of it. It is obviously necessary to learn programming in assembly language. This is why we have elaborated volume 2, which presents the whole elaboration process of an equipment controlled by a microprocessor : an electronic cash register.

Below we present briefly the contens of each chapter.

The prelude consists of three distinct paragraphs. The first one gives a short history of the events which contributed to the appearance of the computer called „von Neumann machine" and it emphasises the indirect contribution of men like John Atanasoff, Alan Turing, Konrad Zuse. In the second paragraph it is constituted the functional structure of a central processing unit, in other words the one of a microprocessor. The last paragraph has the role of introducing the reader in the world of binary numbers — used in all the digital computers.

In **chapter 1 and 2** the hardware structure of the Z80 microprocessor is reveiled. The presentation of the internal structure is done using the funtional blocks defined in Praeludium and shown in images generated by Visible—Z80. We point out that in the block scheme defined by us there appear a few new elements — unpublished up to this time. Their taking into consideration proved to be necessary in order to simulate more faithfully the internal activities of the microprocessor.

Chapter 3 is intended to introduce the term of instruction set. For a better undero standing presentation is done comparatively for two microprocessors. In the same time this chapter tries a rigurous classification of the addressing techniques. It also tries to eliminate some incoherent ambiguities that still exist in different bibliographical sources. At the end of the chapter there is published a possible classification of the Z80 microprocessor's instructions. They are grouped into 12 classes constituted of 98 instruction groups.

Chapter 4 presents the basic working cycles of the processor in case. The reader is guided by means of a double visualization : time diagrams and graphic sequences generated by Visible—Z80. In this way things may be understood even by readers who didn't got familiarized with time diagrams yet.

There is also presented a machine cycle undetailed in the existing literature : the internal machine cycle.

Starting with chapter 5 the book presents the whole instruction set using intensively the above mentioned audiovisual simulator. The text's style is essayistic, also easy to follow by the beginner.

The elements, which couldn't be touched in the graphic sequences or those which result from the author's experience of 10 years' work in this field, are concentrated in remarks numbered from 1 to 67.

Chapter 5 is dedicated to transfer instructions of 8 and 16 bit data. It also presents the data block transfer instructions as : LDIR, LDDR, LDI, LDD.

In **chapter 6** the arithmetic/logic instructions follow, starting with the simple inter-register ones up to those applied to data blocks of variable lenght.

The **next chapter** joins into a single adventure — not in the last instance to break the monotony of some sequences containing instructions of the same type —, instructions belonging to more classes. There are presented the jump instructions, those which call and return from subroutines, as well as those for bit manipulation (BIT, SET, RES), respectively rotate and shift instructions.

The author insists upon the details trying to clear up more and more secrets of Z80's "life". (Have you ever asked yourself : why does the number of clock cycles of an ordinary return instruction differ from those of a conditioned one (i.e. RET NZ and RET) ? If you haven't, you can find the answer in **chapter 7**.

Chapter 8 is dedicated to the input/outout instructions.

Chapter 9 introduces — using again a dual presetation mode (time diagrams and images generated by Visible—Z80) — the special machine cycles. In the same time it introduces some system control instructions. The readers can get aquainted with interupts, with acces to the system buses, with wait states and halt cycles.

The second part of volume 1. presents complementary informations to those described in the first part of the same volume. Therefore the chapters of this part have been noted with letters (A—G)'

Chapter A is the user manual of the Visible—Z80 cassette. There are presented both implementations : the one runnig on Prae and the other one working on aMIC.

The manual describes the 5 major parts of which Visible—Z80 is constituted : monitor, assembler, disassembler, instruction simulator and simulation supervisor.

The reader's initiation into the use of Visible—Z80 is done with the help of images generated by Visible—Z80 itself.

Chapter B is a test report wich describes the activities carried out during the about 1,500 testing hours to which the simulator was submitted before this publication. The chapter ends with a facsimile page containing the image of one of the more than 700 test cards which resulted from testing. The author invites all readers and users to communicate any error that would appear during the use of Visible—Z80.

Chapter C compares the assembly languages of two microprocessors (I8080 and Z80), insisting upon their semantic differences. The chapter's conclusion is that Intel mnemonics are tightly related to the physical reality of the processor, while the Zilog ones, keeping apart from it, offer a greater liberty to the programmer.

Chapter D is exclusively dedicated to those who want to build systems with the help of Z80 microprocessor. There are repeated the detailed time diagrams of basic processor cycles. In order to help the users, the author specifies those 17 timings (from the 53 presented altogether) which must be assured by the hardware designer.

Chapter E contains a card for each of the 98 instructrution groups defined by the author. The informations included in the instruction cards are : generic mnemonic, number of instructions of the considered group, instruction lenght (bytes), instruction duration (clock cycles), the instruction's structure, the addressing modes used, a code memento- with the intention to facilitate the discovery of some code-building rules-, affected flags, as well as references to the pages of volume 2, where these instructions can be seen "in vivo" (in real programs).

From those announced above, results that a proffesionist programmer can find—in this chapter — all the information that will be necessary in his work.

Chapter F contains a list of all the published instructions (696) of the Z80 microprocessor, ordered alphabetically and in increasing order of the codes.

Chapter G insists upon the 458 "hidden instructions" discovered by users, but unpublished by manufacturers.

Volume 2 intends to be a very detailed case-study presenting the complete project on an electronic cash register. So it constitutes to an example of structured programming, in assembly language.

Chapter 10 presents the main elements and methods recommandable to those who want to write quality software.

Chapter 11 contains the technical specification of the product which will be elaborated on the next pages. It specifies, in the same time, the constructive and structural elements and the functional and ergonomic ones of the considered cash register.

The cash register will be provided with 7 special functions, ranging from work session programming to the sale's sythesis of a day, sales sorted, on product items.

The project's result will have to satisfy entirely all requirements formulated in this chapter.

In **chapter 12**, dedicated to the hardware elements the project of man-machine interfaces is elaborated. These interfaces are intended to be poor as hardware. So the possibility appears to elaborate some first software modules.

We mention that in the sound generator's project the implementation's performance is checked. For this purpose it is published a Basic program, too.

The design of hardware structure ends in diagram of the cash register's architecture-Input/output elements are especially pointed out because their knowledge is necessary to launch the main software project.

Before starting the elaboration of pure-software (non hardware related) modules, in **chapter 13** there are defined the main data structures as they derive from the technical specification (chapter 11). Inernal codes, character generators, input/output buffers, BCD registers, parameters and variables appear.

The chapter ends with the draft of the expected data flow in the cash register.

In **chapters 14—16** all the software modules are written which are going to make up the cash register as it was defined in chapter 11.

The followed procedure is the top-down one, starting from the main program and finishing with the last element of detail. The general method used is the following : first of all imposed functions and features are ennumerated followed by the according flowchart The flowchart is afterwards described in a high level language ("pseudo" Pascal). The consi-

dered module's description is finalized by elaborating the equivalent program in assembly language. The program lines are numbered because in the notes that follow, the used techniques will be explained by making references to these line numbers. The notes which follow each written module, oftenly contain suggestions for other possible solutions.

The problems treated in these four chapters cover a large area : BCD arithmetic, syntactic analysis, code conversion, test and service routines, system initialization and supervizing modules.

Chapter 17 contains the whole listing of the created software.

Chapter 18 insits upon elements little tasted by programmers : the elaboration of documentation wich will be able to ensure the software's serviceability even after years.

In **chapter 19** instead of problems and exercises, a work theme is announced. Those who will succed in writing the software modules necessary to implement the given theme, will see the cash register coming to life on the computer's screen and in the same time they can consider themselves formed programmers.

CONTENTS 1 VOLUME

Preface V
Thoughts at the Launching of the First Romanian Book Accompanied by a Program Cassette. Acknowledgements. VI
Publisher's opinion : About the Book and the Author X
General Contents XI
Short English Presentation XII
Detailed Contents of Volume 1 and Volume 2 (English) XV
Detailed Contents of Volume 1 XX

PART I

The Z80 Microprocessor in a Unique Approach Adventures with VISIBLE—Z80

0. Praeludium, Computers and Microprocessors 1
0.1. Considerations upon the Creation of the First Computing Machine with Stored Program 1
0.2. What is a Microprocessor ? 5
0.3. Binary Numbers and their Hexadecimal Representation 10
1. Internal Organization 12
1.1. Registers 13
1.2 Command Unit 28
1 3. Arithmetic Logic Unit 29
1.4. Internal Buses 30
2. External Signals. 31
2.1. Data Bus : D0—D7 32
2.2. Address Bus : A0—A15 32
2.3. Command Bus 33
2.3.1. The Clock 33
2.3.2. Command (Output) Signals 33
2.3.3. Status (Input) Signals 35
3. The Instruction Set 38
3.1. Instruction Format 39
3.2. Addressing Techniques 42
3.2.1. What Are Addressing Techniques ? 43
3.2.2. A Possible Classification 44
3.2.3. Critics upon the Manufacturers Handbook. 52
3.3. Machine Code and Assembly Language 53
3.4. Instruction Classes 55
4. Basic Machine Cycles 57
4.1. Clock Cycle, Machine Cycle, Instruction Cycle 57
4.2. Machine Cycle One : M1 (FETCH) 60

4.3. Memory Read Cycle (READ) . . . 66
4.4. Memory Write Cycle (WRITE) . . . 69
4.4. The Internal Machine Cycle . . . 70

5. Load Instructions . . . 73

5.1. *Adventure 1* : Transfer of 8-Bit Data . . . 75
5.2. Synthesis of Class : LOAD—8 . . . 102
5.3. *Adventure 2* : Transfer of 16-Bit Data . . . 102
5.4. Synthesis of Class : LOAD—16. . . . 112
5.5. *Adventure 3* : Transfer of Data Blocks . . . 113
5.6. Synthesis of Class : LOAD—IDR . . . 119

6. Arithmetic and Logic Instructions . . . 120

6.1. *Adventure 4* : Arithmetic and Logic Instruction Concerning 8-Bit Data . . . 120
6.2. Synthesis of Class : AR/LOG—8 . . . 133
6.3. *Adventure 5* : Arithmetic Instructions Concerning 16-Bit Data . . . 133
6.4. Synthesis of Class : ARIT—16 . . . 143
6.5. *Pseudoadventure 5* : Logic Instructions Concerning Data Blocks . . . 143
6.6. Synthesis of Class : LOG—IDR. . . . 143

7. Other Instructions Classes . . . 144

7.1. *Adventure 6* : Miscellaneous . . . 144
7.2. Synthesis of Class : BiTSR . . . 174
7.3. Synthesis of Class : JUMP . . . 174
7.4. Synthesis of Class : CALL/RET . . . 174
7.5. Synthesis of Class : ROT/SHIFT . . . 175

8. Input/Output Instructions . . . 176

8.1. Input and Output Machine Cycle . . . 177
8.2. *Adventure 7* : Input/Output . . . 178
8.3. Synthesis of Class : IN/OUT . . . 185

9. Special (Machine) Cycles . . . 187

9.1. WAIT Signal and WAIT States . . . 187
9.2. *Adventure 8* : Halt (HALT) . . . 190
9.3. *Adventure 9* : Non Maskable Interrupt (NMI) . . . 198
9.4. *Adventure 10* : Maskable Interrupt and Bus Access (INT, BUSRQ, BUSAK) . . . 211
9.5. Synthesis of Class : SYS . . . 229

PART II

Complements About the Z80

(Instruction Classes and Groups, Published and Hidden Instructions, VISIBLE Z80 : an Audio-Visual Simulator, its Test Procedure and the Cassette Tape's User Manual for two Home Computers Prae and aMIC)

A. VISIBLE—Z80 User Manual . . . 233

A.1.1. Instructions for Prae Users . . . 234
A.1.2. Instructions for aMIC Users . . . 234
A.2. The Enclosed Monitor . . . 236
A.2.1. General Commands . . . 237
A.2.2. Editing Commands . . . 237
A.2.3. Execution Starting Commands . . . 239
A.3. The Graphic Simulator . . . 241
A.3.1. The Supervisor's Static Commands . . . 242
A.3.2. The Supervisor's Dynamic Commands . . . 243
A.4. Examples for Use . . . 244
A.5. Final Recommandations for VISIBLE—Z80 Users . . . 245

B. Test Procedures Used During the Validation of VISIBLE—Z80. 250
(Almost a Test Report) 250
C. Semantic Comparison Between two Related Assembly Languages 257
(I8080 and Z80) . . . . . .
D. Z80 Data Sheets 263
D.1. Static Caracteristics 264
D.2. Dynamic Characteristics 264
E. Z80 Instruction Set Analysis 271
E.1. Instruction Classes and Groups 271
E.2. Instruction Sheets, Essential Features 277
E.3. 8-Bit Transfer Instructions . . . . . LOAD—8 278
E.4. 16-Bit Transfer Instructions LOAD-16 292
E.5. Data Block Transfer instructions . . . . . LOAD—IDR 301
E.6. 8-Bit Arithmetic Logic Instructions . . . . . AR/LOG—8 303
E.7. 16-Bit Arithmetic Instructions . . . . . ARIT—16 336
E.8. Logic Instructions upon Data Blocks . . . . . LOG—IDR 343
E.9. Bit Manipulating Instructions . . . . . BiTSR 346
E.10. Jump Instructions . . . . . JUMP 353
E.11. Subroutine Handling Instructions . . . . . CALL/RET 359
E.12. Rotate and Shift Instructions . . . . . ROT/SHIFT 366
E.13. Input/Output Instructions . . . . . IN/OUT 381
E.14. Command Instructions . . . . . SYS 389
F. List of the 696 Published Z80 Instructions 395
F.1. Alphabetically Sorted List 396
F.2. List Sorted According to Growing Codes. 409
G. The 458 Hidden Z80 Instructions 420
G.1. The "Syndrome". Suggested Mnemonics 421
G.2. The 98 Tested Instructions, Sorted Alphabetically 425

CONTENTS 2 VOLUME

PART. III

Programming the Z80: Case Study: A Cash Register

10. Interregnum. Structured Programming 3
10.1. Software: Art or Profession. Words about Structured Programming . . . 3
10.2. Tools 9
10.2.1. The Flowchart 9
10.2.2. The High Level Language 11
10.2.3. The Assembler 14
11. Product Specification 17
11.1. Structural Specification 17
11.2. Functional Specification 18
11.3. Basic Functions. 20
11.4. Special Functions 22
11.5. Ticket Format 23
12. The Hardware Structure (Architecture) 25
12.1. Components 25
12.1.1. The Keyboard 25
12.1.2. The Display 34
12.1.3. The Printer 39
12.1.4. The Sound Generator. 43
12.2. The Hardware Structure of the Cash Register 56

13. Data Structures 60

13.1. Basic Structures 60

13.1.1. Internal Codes 62
13.1.2. The Display Character Generator: LEDGEN 62
13.1.3. The Printer Character Generator: PRTGEN 63
13.1.4. EPROM Stored Messages 64

13.2. Buffers 65

13.2.1. Keyboard Input Buffer :KEYBUF 66
13.2.2. Display Output Buffer :LEDBUF 66
13.2.3. Printer Output Buffer :PRTBUF 67
13.2.4. Edit Buffer :EDBUF 69
13.2.5. Status Information Dispatcher :SETUPTAB 71

13.3. RAM Registers 72

13.2.1. Expanded BCD Data Register: EBCD 73
13.3.2. BCD Arithmetic Registers 74
13.3.3. General Purpose, RAM Based BCD Registers 74

13.4. Parameters and Variables 75

13.5. Data Flow in the Cash Register (A First Approach) 77

14. Program Implementation (Software) 78

14.1. The Main Loop : MAINLOOP 79

14.1.1. Common Buyer Ticket : BUYTICK 81
14.1.2. FUNC Key Processing : PROCFUNC 83
14.1.3. Work Session Setup : SETUP 86
14.1.4. Erased Buyer Ticket : ERATICK 89
14.1.5. Sorted Item Sums : TOTSORT 91
14.1.6. Total Sales per Day : TOTDAY 93
14.1.7. Synthesis of Sales : SYNTH 94

14.2. Processing Routines 96

14.2.1. Input of a Price : INPRICE 96
14.2.2. Processing of a Price : PRPRICE 98
14.2.3. Erasure of a Price : ERPRICE 99
14.2.4. Preparing Next Price-Input : NXPRICE 100
14.2.5. Item Code Input : INCODE 100
14.2.6. Generating the Implied Item Code : IMPLCODE 101
14.2.7. Succesive FUNC Key Counter : CNTFUNC 102
14.2.8. Operations for the Buyer : OPFORBUY 103
14.2.9. Emit Common Buyer Ticket : EMITBUYTICK 104
14.2.10. Emit Erased Buyer Ticket : EMITERATICK 104
14.2.11. Printing a Ticket from EDBUF : PRTTICK 104
14.2.12. Getting a Wanted Item's Register : SORTADR 107
14.2.13. Update Sum of an Item (+/−) : ADDSORT, SUBSORT . . . 108

14.3. Fixed Point BCD Arithmetic 108

14.3.1. Adding two Numbers : ADDBCD 108
14.3.2. Subtracting two Numbers : SUBBCD 110
14.3.3. Multiplieing two Numbers : MULTBCD 110
14.3.4. Incrementing a Number in Expanded BCD Format : EBCDING 111

14.4. Syntactic Analysis and Code Conversion 113

14.4.1. Analysis and Conversion of Numbers Input from the Keyboard : SYNTAN 113
14.4.2. Packing EBCD Numbers into BCD Format : EBCDBCD 114
14.4.3. Reverse Conversion of Numbers from BCD to Internal Code : BCDCI 116

14.5. Data Transfer 117
14.5.1. Storing a Character in KEYBUF : STORENUM 117
14.5.2. Storing a Character in LEDBUF : DISPNUM 118
14.5.3. Storing a Character in Both KEYBUF and LEDBUF : STORDISP 118
14.5.4. Input Data Alignment : STOREINT 118
14.5.5. Result Display from PRTBUF : DISPSUM 118
14.5.6. Cleaning KEYBUF and LEDBUF : CLBUFDP 119
14.5.7. Filling Memory Area with a Constant : FYLLBYTS 119
14.5.8. Access to TMPBCD : MOVTMP, TMPMOV 119
14.5.9. Dispatching Status Information : TRANSPAR 119
14.5.10. Expanding Messages from EPROM into RAM : EXPAND 120

15. Test Modules 122

15.1. EPROM Test 122

15.2. RAM Test 125
15.2.1. Non Destructive RAM Test (Testing the Contents) 125
15.2.2. Destructive RAM Test (Testing Addresing Circuitry 127

15.3. Display Test 130

15.4. Printer Test 130

16. System Init and Supervising 134

16.1. Cold Start : CSTART 135

16.2. Warm Start : WSRT 136

16.3. Protection Against Accidental Power Failure : DROPOUT 138

17. Program Listing 141

18. Project Memento 217

18,1. RAM Memory Map 217

18.2. Data Flow in the Cash Register (Final Approach) 219

18.3. Hierarchy of Written Routines 219

18.4. Written Routine List 221

19. Instead of Problems and Exercises 224

19.1. Theme Enunciation 224

19.2. Working Guide 225
19.2.1. Imposed Tasks 227
19.2.2. Tasks at Choice 227

19.3. ASCII Code Table 228

19.4. Helpful Routines in Your Computer: Prae and aMIC 229

References 232

Cuprins detaliat

Volumul 1

Prefață V

Cuvînt înainte : Gînduri la editarea de către Editura Tehnică a primei cărți românești însoțite de o casetă cu program de calculator VI
Opinia editorului : Despre carte și despre autor X
Cuprins general XI
Prezentarea cărții în limba engleză XII
Cuprins detailat al cărții (l. engleză) XV
Cuprins detaliat al volumului 1 XX

Partea I

MICROPROCESORUL Z80 : o prezentare inedită. Simularea funcționării lui Z80, cu și fără casetă magnetică. Aventuri cu programul de simulare VISIBLE—Z80

Cap. 0. **Praeludium. Calculatoare și microprocesoare** 1

0.1. Retrospectivă asupra apariției primului calculator cu program memorat . . 2
0.2. Ce este microprocesorul ? 5
0.3. Numere binare și reprezentarea lor hexazecimală 10

Cap. 1. **Structura internă** 11

1.1. Regiștri 13
1.2. Circuitul de comandă 28
1.3. Unitatea aritmetică/logică 29
1.4. Magistralele interne 30

Cap. 2. **Semnalele externe** 31

2.1. Magistrala de date : D0 — D7 32
2.2. Magistrala de adrese : A0 — A15 32
2.3. Magistrala de comenzi 33
2.3.1. Semnalul de ceas (tact) 33
2.3.2. Semnale de comandă 33
2.3.3. Semnale de stare 35

Cap. 3. **Setul de instrucțiuni** 38

3.1. Formatul instrucțiunilor 39
3.2. Tehnici de adresare 42
3.2.1. Ce sînt tehnicile de adresare ? 43
3.2.2. O clasificare posibilă 44
3.2.3. Critica documentației de firmă 52
3.3. Codul mașină și limbajul de asamblare 53
3.4. Clasificarea instrucțiunilor 55

Cap. 4. **Cicluri de bază** 57

4.1. Ciclul de tact, ciclul mașină, ciclul instrucțiune 57
4.2. Ciclul mașină M1 (FETCH) 60
4.3. Ciclul mașină de citire din memorie (READ) 66
4.4. Ciclul mașină de scriere în memorie (WRITE) 69
4.5. Ciclul mașină intern 70

Cap. 5. **Instrucțiuni de transfer** 73

5.1. *Aventura 1* : transferuri de 8 bit 75
5.2. Sinteza clasei : LOAD—8 102
5.3. *Aventura a 2-a* : transferuri de 16 bit 102
5.4. Sinteza clasei : LOAD—16. 112
5.5. *Aventura a 3-a* : transferul blocurilor de date 113
5.6. Sinteza clasei : LOAD—IDR 119

Cap. 6. **Instrucțiuni aritmetice și logice** 120

6.1. *Aventura a 4-a* : aritmetică și logică pe 8 bit 120
6.2. Sinteza clasei : AR/LOG—8 133
6.3. *Aventura a 5-a* : aritmetica pe 16 bit 133
6.4. Sinteza clasei : ARIT—16 143
6.5. Pseudoaventură. Instrucțiunile logice pe blocuri de date 143
6.6. Sinteza clasei : LOG—IDR. 143

Cap. 7. **Instrucțiuni diverse** 144

7.1. *Aventura a 6-a* : de toate 144
7.2. Sinteza clasei : BiTSR 174
7.3. Sinteza clasei : JUMP 174
7.4. Sinteza clasei : CALL/RET 174
7.5. Sinteza clasei : ROT/SHIFT 175

Cap. 8. **Instrucțiuni de intrare/ieșire** 176

8.1. Ciclurile mașină de intrare și de ieșire 177
8.2. *Aventura a 7-a* : intrare/ieșire 178
8.3. Sinteza clasei : IN/OUT 185

Cap. 9. **Cicluri speciale** 187
9.1. Semnalul și stările WAIT 187
9.2. *Aventura a 8-a* : oprirea (HALT) 190
9.3. *Aventura a 9-a* : întreruperea nemascabilă (NMI) 198
9.4. *Aventura a 10-a* : întreruperile mascabile și accesul la magistrale (INT, BUSRQ, BUSAK) 211
9.5. Sinteza clasei : SYS 229

Partea a II-a

Complemente privind microprocesorul Z80: clase de instrucțiuni, instrucțiuni declarate, instrucțiuni descoperite, comparația limbajelor microprocesoarelor Z80 și I8080, programul de simulare pe mediu magnetic (caseta VISIBLE—Z80), procedura sa de testare și manualul de utilizare a casetei pe calculatoarele personale PRAE și aMIC

A. *Manualul de utilizare a casetei* cu produsul program *VISIBLE—Z80* și cu programul proiectului casei de marcat electronice 233

A.1. Încărcarea și lansarea programului 234
A.1.1. Comenzi pentru calculatorul PRAE—M. 234
A.1.2. Comenzi pentru calculatorul aMIC. 236

A.2. Comenzile monitorului încorporat 237
A.2.1. Comenzi generale 237
A.2.2. Comenzi de editare 239
A.2.3. Comenzi de lansare în execuţie 241
A.3. Simulatorul grafic propriu-zis 242
A.3.1. Comenzile statice ale supervizorului 243
A.3.2. Comenzile dinamice ale supervizorului 244
A.4. Exemple de lucru în ambianţa VISIBLE—Z80 245
A.5. Recomandări pentru utilizarea pachetului VISIBLE—Z80 249

B. *Procedura de testare de simulare pentru validarea programului VISIBLE—Z80* . . . 250

B.1. Obiectivul propus şi metoda folosită. 250
B.2. Rezultatele testării 252
B.3. Răspunsul elaboratorilor 253
B.4. Probleme nerezolvate 254

C. *Comparaţia semantică a limbajelor de asamblare ale microprocesoarelor I8080 şi Z80* 257

D. *Datele de catalog ale microprocesorului Z80* 263
D.1. Parametri statici 264
D.2. Parametri dinamici 264

E. *Analiza setului de instrucţiuni ale microprocesorului Z80* 271

E.1. Clase şi grupe de instrucţiuni 271
E.2. File de instrucţiuni. 277
E.3. Instrucţiuni de transfer de 8 bit LOAD—8 278
E.4. Instrucţiuni de transfer de 16 bit LOAD—16 292
E.5. Instrucţiuni de transfer de blocuri de date LOAD—IDR 301
E.6. Instrucţiuni aritmetice logice pe 8 bit AR/LOG—8 303
E.7. Instrucţiuni aritmetice pe 16 bit ARIT—16 336
E.8. Instrucţiuni logice pe blocuri de date LOG—IDR 343
E.9. Instrucţiuni orientate pe bit BiTSR 346
E.10. Instrucţiuni de salt JUMP 353
E.11. Instrucţiuni de apel şi revenire din subrutine CALL/RET 359
E.12. Instrucţiuni de rotire şi deplasare ROT/SHIFT 366
E.13. Instrucţiuni de intrare/ieşire IN/OUT 381
E.14. Instrucţiuni de comandă SYS 389

F. Lista *celor 696 instrucţiuni declarate ale microprocesorului Z80* 395

F.1. Lista în ordine alfabetică a mnemonicelor 396
F.2. Lista de ordine creşcătoare a codurilor 409

G. *Cele 458 instrucţiuni ascunse ale microprocesorului Z80* 420

G.1. Prezentarea fenomenului. Mnemonici propuse. 421
G.2. 98 instrucţiuni testate, în ordinea alfabetică a mnemonicelor 422

Partea I

MICROPROCESORUL Z80: O PREZENTARE INEDITĂ. SIMULAREA FUNCŢIONĂRII CU ŞI FĂRĂ CASETĂ. AVENTURI CU PROGRAMUL DE SIMULARE VISIBLE–Z80.

A.2. Comenzile monitorului încorporat 237
A.2.1. Comenzi generale 237
A.2.2. Comenzi de editare 239
A.2.3. Comenzi de lansare în execuție 241
A.3. Simulatorul grafic propriu-zis 242
A.3.1. Comenzile statice ale supervizorului 243
A.3.2. Comenzile dinamice ale supervizorului 244
A.4. Exemple de lucru în ambianța VISIBLE—Z80 245
A.5. Recomandări pentru utilizarea pachetului VISIBLE—Z80 249

B. Procedura de testare de simulare pentru validarea programului VISIBLE—Z80 . . . 250

B.1. Obiectivul propus și [illegible] 250
B.2. Rezultatele testării 252
B.3. Răspunsul elaboratorilor 253
B.4. Probleme [illegible] 254

C. Comparație semantică a simbolurilor de mnemonice ale microprocesoarelor 8080 și Z80 257

D. Datele de catalog ale microprocesorului Z80 263
D.1. Parametri [illegible] 264
D.2. Parametri dinamici 264

E. Analiza [illegible] 275

E.1. Clase și grupe de instrucțiuni 275
E.2. [illegible] instrucțiunii 277
E.3. Instrucțiuni de transfer de 8 bit LOAD—8 278
E.4. Instrucțiuni de transfer de 16 bit LOAD—16 292
E.5. Instrucțiuni de transfer de blocuri de date LOAD—IDR 301
E.6. Instrucțiuni aritmetice logice pe 8 bit ARILOG—8 303
E.7. Instrucțiuni aritmetice pe 16 bit ARIT—16 336
E.8. Instrucțiuni logice pe blocuri de date LOG—IDR 343
E.9. Instrucțiuni orientate pe bit BITER 346
E.10. Instrucțiuni de salt JUMP 353
E.11. Instrucțiuni de apel și revenire din subrutine CALL/RET 359
E.12. Instrucțiuni de rotire și deplasare ROT/SHIFT 366
E.13. Instrucțiuni de intrare/ieșire IN/OUT 381
E.14. Instrucțiuni de comandă SYS 389

F. Lista celor 696 instrucțiuni declarate ale microprocesorului Z80 395

F.1. Lista în ordine alfabetică a mnemonicelor 396
F.2. Lista de ordine crescătoare a codurilor 409

G. Cele 458 instrucțiuni ascunse ale microprocesorului Z80 420

G.1. Prezentarea fenomenului. Mnemonici propuse. 421
G.2. 98 instrucțiuni testate, în ordinea alfabetică a mnemonicelor 432

0

PRAELUDIUM

Calculatoare și microprocesoare

„Muncește cu sîrg și cu har
Cum astrul se mișcă pe cer
solitar"•

József Attila

0.1. Retrospectivă asupra apariției primului calculator cu program memorat.

Am trăit o reală surpriză, cînd la Conferința **"Children in an Information Age"** desfășurată sub egida UNESCO în 1985 la Varna, un domn bătrîn, pe nume **John Atanasoff** a fost prezentat drept autorul primului calculator electronic modern. Pînă în acel moment „paternitatea" atribuită lui **John von Neumann** părea de necontestat, ca și informația conform căreia primul calculator electronic din lume a fost ENIAC, iar numele domnului John Atanasoff nu fusese amintit în legătură cu acest proiect. Întrucît afirmația fusese făcută în plen, în fața a sute de specialiști întruniți din toate colțurile lumii, ea nu putea fi lipsită de adevăr. Iată de ce am simțit nevoia de a întreprinde cîteva cercetări bibliografice pentru elucidarea acestei întrebări cheie.

S-a demonstrat că John von Neumann nici nu știuse de proiectul ENIAC pînă la demararea lucrărilor de realizare fizică a proiectului deja elaborat. Apoi am aflat că mai sînt cel puțin o duzină de revendicări formulate din diverse părți ale globului, legate de numele unor specialiști remarcabili, din rîndul cărora îi amintim pe **Konrad Zuse, Alan Turing, John W. Mauchly, J. Presper Eckert jr.**

Concluziile la care am ajuns fiind interesante, am decis includerea sintezei lor în prezenta carte, dedicată aproape exclusiv unității centrale a calculatorului actual — microprocesorul —

La mijlocul anilor '30 tehnica electronică (bazată pe tuburi) a ajuns la un nivel de dezvoltare care i-a permis aplicarea în domenii din ce în ce mai diverse, depășind limitele radiofoniei.

În mai multe centre universitare din Europa și America s-au demarat cercetări pentru îmbunătățirea pe cale electronică a performanțelor calculatoarelor electromecanice deja existente.

În primăvara anului 1943 la Institutul **Moore** al Universității Pennsylvania, John W. Mauchly propunea realizarea unui calculator digital complet electronic.

• În românește de Mihai Beniuc·

Iată cum prezenta împreună cu J. G. **Brainerd** şi J. P. Eckert modelul calculatorului propus :

„După cum aminteam, pentru obţinerea rezultatelor dorite calculatorul electronic va folosi principiul contorizării. De aceea el nu este altceva decît analogul electronic al maşinilor mecanice de adunare, înmulţire şi împărţire care se fabrică în zilele noastre în scopul efectuării unor operaţii aritmetice. Structura constructivă a calculatorului electronic permite însă realizarea unor comunicaţii între diversele sale părţi componente, putîndu-se realiza astfel cicluri de funcţionare, care vor permite, în limita posibilităţilor, calcularea oricăror ecuaţii diferenţiale, prin iteraţii succesive.

Rezultatul unei operaţii, de exemplu cel al unei înmulţiri, poate fi folosit nemijlocit în cadrul altor calcule succesive, conform modulului impus de ecuaţiile ce „guvernează" problema de rezolvat. Aceste numere vor putea fi transferate în caz de nevoie dintr-o componentă a sistemului în alta, fără ca în prealabil să fim nevoiţi a le copia manual pe hîrtie, sau în componenta destinaţie, aşa cum ar fi necesar dacă rezolvarea iterativă s-ar face cu calculatoare ordinare".

Trebuie să amintim faptul că J. W. Mauchly a avut consfătuiri cu John Atanasoff de la Institutul Superior de Stat din Iowa, care experimentase deja unele modalităţi de rapidizare pe cale electronică a maşinilor de calcul obişnuite.

Astfel, cînd la data de 1 iulie 1943 se lansa proiectul **ENIAC** (Electronic Numeric Integrator and Calculator) existau de acum cîteva calculatoare dotate cu facilităţi electronice.

Proiectul electronic al calculatorului ENIAC s-a terminat în aproximativ **1 an.** La realizarea lui a participat un grup de ingineri deosebit de dotaţi, conduşi de J. Presper Eckert jr. Iată numele cîtorva care au jucat un rol important în derularea acestei munci creatoare desfăşurate zi şi noapte de-a lungul celor aproape 3 ani (ultimii doi dedicaţi realizării fizice) : John W. Mauchly, Arthur W. Burks, Joseph Chedaker, Chuan Chu, John H. Davis, Adele K. Golstine, Harry Huskey, T. Kite Sharpless şi Robert Show.

La data de 15.02.1946 ENIAC — deci primul calculator complet electronic, se predă beneficiarului. ENIAC — o construcţie gigant (conţinea aproximativ 20.000 tuburi electronice) vehicula numere zecimale. În memoria sa nu se puteau înmagazina decît 20 de numere cu 10 cifre semnificative. ENIAC nu a fost un calculator programabil în accepţiunea de astăzi a cuvîntului, ci a permis doar realizarea unor bucle iterative pentru înlănţuirea operaţiilor aritmetice de bază.

Cu toată lipsa sa de fiabilitate — datorită duratei de viaţă limitate a tuburilor electronice folosite — şi a memoriei sale limitate, ENIAC a reuşit să învingă toate reţinerile formulate la adresa calculatorului electronic, prin viteza de calcul ridicată.

Comparînd performanţele lui cu calculatorul electromecanic **Mark I** se constată că acesta din urmă rezolva ecuaţia diferenţială parţială formulată de Neumann, în 80 ore, pe cînd folosind ENIAC problema se rezolva în aproximativ 30 minute din care 28 minute se consumau pentru perforarea cartelelor necesare memorării temporare a valorilor intermediare de calcul. (Calculatorul Mark I realizat sub conducerea profesorului **Howard Aiken** la Universitatea **Harvard** a fost un calculator electromecanic ce conţinea aproximativ 760.000 piese. Acesta a fost calculatorul care funcţionînd ani în şir fără eroare, a întemeiat supremaţia mondială a firmei **IBM,** înfiinţată încă din 1911 sub numele de **Computing Tabulation Company.** Printre membrii fondatori îl găsim şi pe **Hermann Hollerith,** cel care a elaborat cartelele perforate care se folosesc pînă în ziua de astăzi).

John von Neumann, care trăia în orașul în care se elabora proiectul ENIAC, n-a fost cooptat decît în momentul în care lucrările au ajuns în faza execuției fizice a calculatorului.

Iată deci, deși n-a participat la elaborarea primului calculator complet electronic, Neumann urma să devină *prin contribuția sa teoretică esențială în fundamentarea următoarelor proiecte*, cel care avea să fie privit drept fondatorul calculatoarelor moderne.

Cooptarea lui Neumann s-a făcut în vara anului 1944. Fiind deosebit de interesat în utilizarea unui calculator rapid pentru rezolvarea unor ecuații de tipul celei amintite, el s-a implicat profund în proiectul ce se derula, remarcînd cîteva insuficiențe ale acestuia. Ideea fundamentală care s-a cristalizat în perioada 21 aug. — 2 septembrie 1944, în cadrul discuțiilor cu autorii proiectului ENIAC a fost aceea de a înmagazina în memoria calculatorului pe lîngă datele calculate, însuși programul, automatizîndu-se astfel întreaga activitate a calculatorului.

Nu se poate ști exact cine din cei participanți la discuții a formulat această idee. Neumann a scris-o pe hîrtie, dezvoltînd-o și detailînd-o la nivelul unui proiect logic care avea să fundamenteze structura constructivă și funcțională a calculatorului electronic. Cert este că Neumann a fost catalizatorul discuțiilor, și faptul că el cunoscuse lucrările lui Alan Turing.

Alan Turing un tînăr matematician englez care și-a realizat lucrarea de doctorat la Universitatea din **Princeton**, a elaborat o lucrare care poate fi considerată esențială în fundamentarea matematică a științei calculatoarelor. Numele lui se vehiculează mai rar în analele istoriei calculatoarelor electronice, deoarece el nu a acceptat postul de asistent al lui John von Neumann, ofertă care probabil i-ar fi deschis calea către o strălucită afirmare profesională, ci a preferat să se întoarcă în țara sa natală pentru a se angaja la Ministerul Afacerilor Externe, aducîndu-și pe această cale aportul la lupta împotriva pericolului fascist iminent.

Alan Turing a elaborat conceptul numerelor „calculabile", termen care-i aparține. Numerele calculabile sînt acele numere reale a căror parte zecimală se poate determina printr-un număr finit de iterații. Marele lui merit constă în aceea, că a demonstrat faptul că există un automat universal care poate determina numerele calculabile și a elaborat modelul unui asemenea automat.

Automatul lui Turing este desigur unul matematic, abstract. Iată esența lui :

a) automatul are un număr de n stări finite ;

b) automatul se află în orice moment într-o stare i, unde $1 \leqslant i \leqslant n$ urmînd ca în momentul următor să treacă într-o altă stare j ($1 \leqslant j \leqslant n$).

c) fiecare stare este caracterizată prin următoarele informații :

1. Valoarea caracteristică e_i (o valoare „curentă" a numărului de calculat)
2. Funcția f_i care se va aplica valorii caracteristice e_i pentru a obține următoarea valoare caracteristică e_j
3. Deplasamentul d_{ij} care va trebui aplicat numărului i pentru a se determina starea j : $j = i + d_{ij}$

Prin aceste elemente automatul universal este complet definit. Încercînd să concretizăm modelul de mai sus, ne vom închipui o bandă de hîrtie pe care sînt marcate, de exemplu prin găuri, n cîmpuri de date aferente celor n stări distincte ale automatului. (A nu se confunda cu banda de hîrtie perforată folosită curent în tehnica de calcul a anilor '70).

Fiecare cîmp de date conţine cele 3 mărimi din informaţia de stare, enumerate la pct. c) şi o zonă de manevră pe care se poate şterge şi înregistra o nouă valoare numerică. Automatul fizic va fi apt să deplaseze banda înainte şi înapoi cu un număr de poziţii d_{ij}. El îşi va desfăşura activitatea permanent pe baza informaţiilor citite de pe bandă.

Să presupunem că automatul se află în starea i (în dreptul cîmpului de date i). Va citi valoarea caracteristică e_i, îi aplică funcţia f_i, citită din acelaşi cîmp de date, urmînd ca apoi să citească deplasamentul d_{ij}, cu care va trebui să mişte banda înainte sau înapoi pentru a ajunge în starea următoare j.

Acesta este modelul automatului universal. Dacă specificăm mărimile e_i, f_i, d_{ij} unde $i, j \in [1, n]$ atunci s-a definit un automat dedicat, care va rezolva problema impusă. Ce altceva este automatul universal al lui Turing, decît modelul matematic al unui calculator cu program memorat ?

Pentru a se elimina deficienţele constatate ale calculatorului ENIAC, cu aproximativ 17 luni înainte de realizarea lui fizică s-a demarat proiectul unui nou calculator, **EDVAC** (13 septembrie 1944).

În cadrul acestui proiect, Neumann a elaborat o lucrare intitulată „**First draft of a report on the EDVAC**", (30 iulie 1945), o lucrare de bază în care propune structura funcţională a calculatorului, circuitele sale logice, cum şi setul de instrucţiuni.

Ilustrînd modul de utilizare (programare) a acestui set de instrucţiuni, lucrarea conţine şi implementarea unui algoritm de clasificare — sortare.

O dată cu terminarea războiului, grupul EDVAC s-a destrămat. Cei care au rămas la Princeton au abordat la Institute for Advanced Study (IAS) proiectul unui nou calculator ce urma să fie un calculator *binar*, inspirîndu-se în realizarea memoriei, din tehnica tuburilor cinescop, şi avînd o unitate de prelucrare paralelă.

Pentru prima dată s-au separat colective de lucru pentru *proiectarea logică* şi pentru cea *hardware*. Alături de Neumann, care a condus proiectul logic va trebui să-i amintim şi pe **Herman H. Goldstine, Julian Bigelow** şi **James Pomerene.**

Proiectul s-a finalizat, rezultatul fiind calculatorul care poartă diverse nume : **IAS, Princeton, maşina Neumann, MANIAC,** sau **Johnniac.**

Pe lîngă impunerea sistemului de numeraţie binar, şi pe lîngă memorarea programului s-a constituit şi structura logică a calculatorului electronic digital.

Se disting de acum părţile funcţionale majore :

— **unitate centrală**
— **memorie**
— **dispozitive de intrare/ieşire**

Unitatea centrală va cunoaşte un set de instrucţiuni, codificate binar, care vor fi citite rînd pe rînd din memoria de lucru. Unitatea centrală va decodifica şi executa instrucţiunea curentă, urmînd să treacă controlul la instrucţiunea următoare.

Aceasta este maşina de referinţă pentru următoarele 4 decenii ale tehnicii de calcul.

Amintind că prima demonstraţie de teleprelucrare de date s-a făcut în 1940 (Bell Telephone Laboratories — **Stibitz** şi **Andrew**) între **Dartmouth** şi **New York** folosind o maşină de telex şi un calculator electromecanic, încheiem prezentarea istoriei apariţiei calculatoarelor electronice cu program memorat. Va trebui să constatăm, că aidoma multor altor descoperiri tehnice şi ştiinţifice, şi apariţia acestor calculatoare este rodul unui larg şi îndelungat proces de creaţie umană.

Nu putem trece cu vederea contribuțiile anterioare : **Descartes** (1647), **Pascal** (1650), **Leibnitz** (1673) și **Babbage** (1822, 1833) ș.a.

În anii 1940 crearea calculatorului electronic modern „plutea în aer", ea fiind iminentă. John von Neumann a fost cel care prin asimilarea rezultatelor parțiale, mai de mult sau mai de curînd create, a reușit să elaboreze o sinteză calitativ nouă.

0.2. Ce este microprocesorul ?

În anul 1964, **Gordon E. Moore** — director, la vremea respectivă, la Centrul de cercetări al firmei **Fairchild** — a remarcat faptul că de la apariția primului tranzistor planar (1958 — **Jean Hoerni**, la Fairchild) numărul tranzistoarelor planare create pe aceeași structură de siliciu s-a dublat anual. În acest moment el a lansat ideea, care de atunci îi poartă numele, afirmînd că acest ritm de dezvoltare se poate menține de-a lungul a cîtorva decenii. Conform „Legii" lui Moore, după 22 de ani o structură de circuit integrat va fi putut conține 2^{22}, adică mai mult de 4 milioane de elemente de circuit electronice. Previziunea s-a adeverit. În anul 1986 s-au realizat primele circuite integrate de memorie, avînd capacitatea de 4 Mbit (4 milioane bit).

Dacă o suprafață de aproximativ 2 cm^2 trebuie să conțină 4 milioane de elemente distincte, atunci dimensiunea medie a unui element va fi de 50 μm^2. Grosimea traseelor de legătură între elemente este mai mică de 1 μm.

Ținînd cont de aceste exigențe, cerințele tehnologice impun anumite „reguli de joc" extrem de severe. Să amintim doar două din ele. Puritatea materialelor chimice care se folosesc în procesul de elaborare a structurii de circuit integrat trebuie să depășească 99,9999999%. În încăperile de lucru, numărul particulelor de praf a căror diametru este mai mare decît 1 μm nu are voie să depășească limita de 2700 particule/m^3 de aer.

Pentru a oferi un termen de comparație, amintim faptul că cerințele impuse celor mai sterile săli de operație din clinici se limitează la nivelul de 270.000 particule/m^3.

Rigoarea unor asemenea cerințe tehnologice este răsplătită prin dezvoltarea exponențială a acestei ramuri industriale, în care ponderea inteligenței încorporate în produse depășește cu multe ordine de mărime pe cea a materiilor prime și a energiei consumate în procesul de fabricație. Astfel s-a ajuns ca după 40 de ani de existență a industriei de calculatoare electronice, cel mai simplu calculator personal, (care încorporează un microprocesor cu valoare aproximativ 6 $ și al cărui preț total nu depășește valoarea de 150$) să depășească net, la toate capitolele de performanță, calculatoarele electronice din prima generație. Dacă ar fi să comparăm un calculator personal, bun de larg consum, cu primul calculator electronic, ENIAC, constatăm că obiectul nostru este de 40 de ori mai rapid, capacitatea sa de memorie este de 300 de ori mai mare, fiabilitatea sa este mai mare de mii de ori. Prețul scade de 20.000 ori, iar gabaritul cu un factor de 30.000. Pe cînd ENIAC consuma o putere echivalentă cu cea a unei locomotive, întregul sistem microcalculator (calculator, televizor, casetofon) nu necesită mai multă energie electrică decît un bec de 100 W.

Lumea microelectronicii este o lume fabuloasă. Microprocesorul este, chiar dacă nu cel mai complex, dar în mod cert cel mai reprezentativ element al acestei lumi.

Structura de bază a unui microprocesor virtual

Microprocesoarele nu sînt altceva decît unități centrale de calculator (**CPU** — Central Processing Unit). încorporate într-o singură capsulă de circuit integrat.

Ele vor citi instrucțiunile unui program dintr-un bloc de memorie, le vor decodifica și vor executa comenzile formulate în însuși codul instrucțiunii.

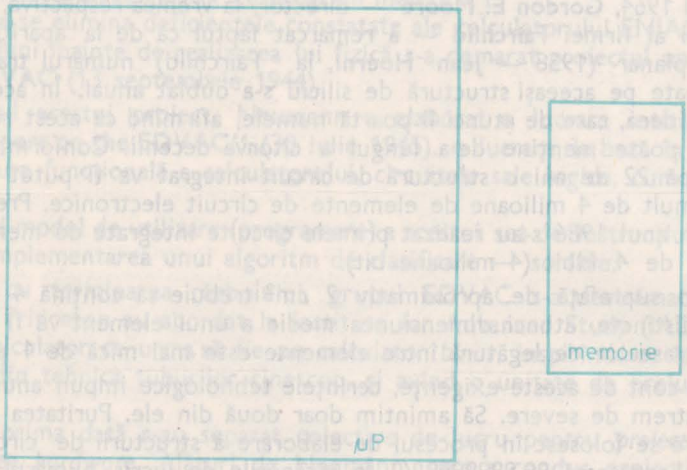


Fig. 0.1.

Vom încerca să umplem treptat blocul funcțional gol din fig. 0.1. cu elementele strict necesare pentru funcționarea unei unități centrale, constituind pe parcursul prezentului paragraf o posibilă structură de microprocesor.

Pentru a citi din blocul de memorie externă codul instrucțiunii ce urmează a fi executată, microprocesorul va trebui să genereze o adresă pe care o va pune la dispoziția memoriei, pînă cînd din celula selectată pe baza acestei adrese va apare data cerută. Pentru a putea „menține" starea liniilor de adresă pe durata întregii operații de citire, microprocesorul va trebui să posede un element memorator intermediar, pe care-l vom numi **registrul tampon de adrese AB (Address Buffer).**

Informația codificată, citită din memorie o vom depune temporar, de asemenea într-un registru intermediar. Fie numele acestuia **registru tampon de date DB (Data Buffer).**

Liniile electrice pe care se va genera cuvîntul binar de adresă le vom numi **magistrală de adrese ABUS (Address Bus),** iar pe cele dedicate datelor citite/scrise în memorie, **magistrală de date DBUS (Data Bus).** Completînd modelul inițial vid, cu aceste elemente, obținem structura din fig. 0.2.

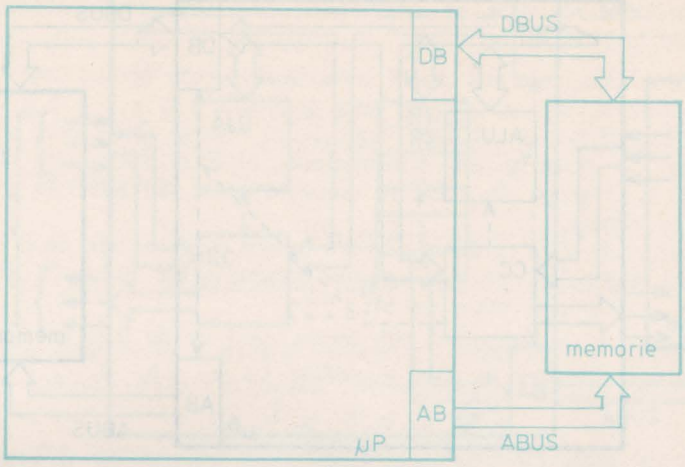


Fig. 0.2.

Să presupunem că instrucțiunea recent citită din memorie și depusă temporar în registrul de date are următoarea semnificație : *„Citește conținutul celulei de memorie a cărei adresă este cu 3 mai mare decît adresa curentă (cea din AB), adaugă la această valoare 5 și rescrie rezultatul în aceeași celulă de memorie."*

Pentru a efectua această instrucțiune microprocesorul are nevoie de o unitate aritmetică : cu ajutorul acesteia se va putea calcula noua adresă de memorie și se va putea efectua adunarea cerută. După cum am amintit deja, în aritmetica binară operațiile aritmetice pot fi descrise cu ajutorul unor funcții logice. Este normal ca aceste circuite să fie utilizate și pentru efectuarea unor operații logice propriu-zise. De aici rezultă și numele utilizat : **unitate aritmetică/logică ALU (Arithmetic Logic Unit).**

Pentru a putea executa cele formulate în enunțul comenzii, microprocesorul va trebui să fie dotat și cu o **unitate de comandă.**

Ea este aceea care va diseca problema, „spărgînd-o" în pași elementari și va programa execuția secvențială în timp a tuturor manevrelor necesare pentru a duce la bun sfîrșit misiunea ce i-a fost încredințată : generează semnalele de comandă pentru întregul sistem, dirijează fluxul de date, corelează viteza de lucru a unității centrale cu timpul de acces al memoriei etc. Unitatea de comandă poartă diverse nume : **CCU — Computer Control Unit, TCU — Timing and Control Unit, SQ — SeQuencer sau CC — Command Circuit.** Noi vom alege ultimul nume. Activitatea unității de comandă este pilotată de un semnal de ceas, avînd frecvența de ordinul MHz ($1M=10^6$).

Semnalele electrice prin care microprocesorul va da comenzi de execuție către memorie și celelalte componente din sistem, le vom numi **semnale de comandă.** Semnalele prin care el culege informații privind starea diverselor componente din sistem, le vom numi **semnale de stare.** Introducînd elementele nou definite în modelul considerat, obținem structura din fig. 0.3.

Să continuăm construcția. Dacă următoarea instrucțiune va folosi rezultatul instrucțiunii precedente pentru a efectua o nouă operație aritmetică, atunci valoarea calculată în prealabil trebuie citită din nou din memorie. Acest acces suplimentar la memorie poate fi economisit, dacă în interiorul microprocesorului

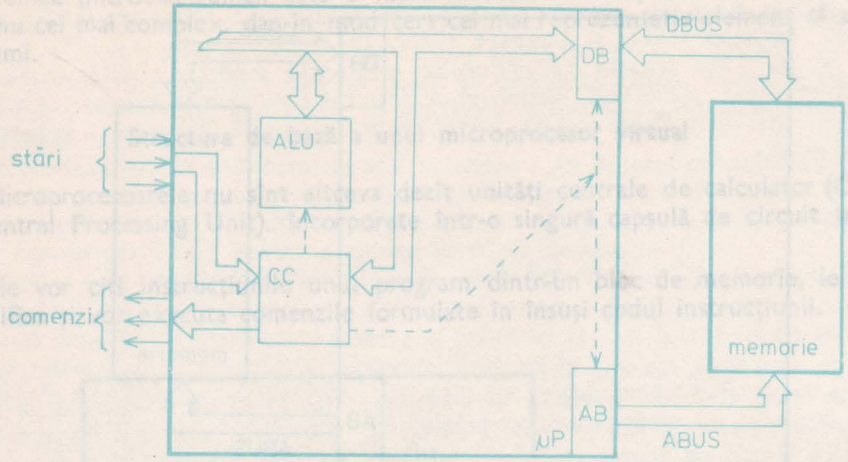


Fig. 0.3.

vom prevedea cîteva elemente de memorare în care să se poată înmagazina temporar date sau adrese de memorie. Aceste elemente de memorie ale microprocesorului le vom numi **regiştri.** Unii din aceşti regiştri vor fi folosiţi în scopuri dedicate, cum ar fi urmărirea execuţiei secvenţiale a instrucţiunilor din memorie. Registrul tampon de adrese (AB) nu poate fi folosit în acest scop, fiindcă aşa cum s-a văzut în exemplele prezentate, conţinutul lui va trebui să se modifice, eventual de mai multe ori pe parcursul execuţiei unei instrucţiuni (cazul în care instrucţiunea în curs trebuie să efectueze accese la memorie, pentru a citi sau a scrie date în ea). Este deci nevoie de un registru în care să se poată genera şi păstra nealterată adresa de memorie a următoarei instrucţiuni de executat. Folosind acest registru, microprocesorul va putea continua — după terminarea instrucţiunii în curs — execuţia secvenţei de instrucţiuni prevăzute, care astfel se constituie într-un program.

Numele acestui registru special este **contor de program PC (Program Counter).**

Într-un alt registru special vom putea păstra informaţii referitoare la natura rezultatului unei operaţii aritmetice : număr negativ sau pozitiv (eventual egal cu zero), număr par sau impar etc. În acest registru, fiecărui atribut considerat i se va rezerva un bit. Vom numi aceşti biţi, biţi de condiţie, iar registrul îl vom numi registrul **indicatorilor de condiţie F (Flag** — steguleţ, fanion). Avem nevoie de acest registru pentru a putea defini instrucţiuni de salt condiţionat. Să ne imaginăm următoarea instrucţiune : „Dacă rezultatul operaţiei precedente este un număr negativ, atunci continuă execuţia programului cu instrucţiunea locată la adresa de memorie următoare (valoarea actuală a contorului de program, PC) ; dacă numărul este pozitiv, continuă execuţia la adresa egală cu valoarea conţinută în primul registru de uz general". Această instrucţiune va verifica valoarea „indicatorului de semn" şi va ramifica execuţia programului în funcţie de valoarea acestui bit.

Să introducem şi regiştri în schema bloc din fig. 0.3. Cu aceasta putem considera încheiată definirea principalelor structuri funcţionale ale microprocesorului.

Pentru efectuarea transferurilor interne de date, microprocesorul va fi prevăzut şi cu o **magistrală internă de date,** magistrală a cărei „lăţime" (număr de linii distincte) depinde de tipul microprocesorului în cauză.

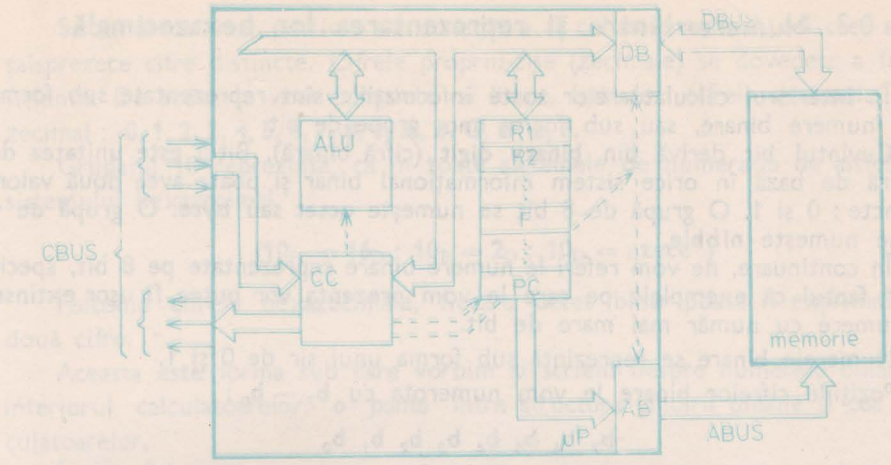


Fig. 0.4.

Privind din exterior, la terminalele microprocesorului regăsim semnalele sale de comunicație cu sistemul constituit în jurul lui : magistrala de adrese, magistrala de date și magistrala de comenzi.

La magistralele de date și de comenzi pot fi cuplate **circuite de intrare/ieșire (I/O)** care stabilesc legătura cu lumea externă : interfața om-mașină și accesul la memorii de masă, nevolatile. Dacă veți completa fig. 0.4 cu cîteva dispozitive de intrare/ieșire, veți obține schema bloc a hardware-ului minimal al unui microcalculator.

Ideea de magistrală unică, „plimbată" la toate elementele funcționale din sistem — inclusiv la cele de intrare/ieșire — este o caracteristică constructivă de bază a microcalculatoarelor. Amintim aici, că unitățile de calcul mai performante efectuează operațiile de intrare/ieșire prin așa numitele canale de I/O, a căror complexitate poate depăși uneori complexitatea unui microcalculator. Ele sînt capabile să efectueze transferul de date între calculator și mediul extern în mod independent, fără ca prin aceasta să perturbe cîtuși de puțin derularea unui program de către unitatea centrală.

„Lungimea" (numărul de bit a regiștrilor interni ai microprocesorului) se corelează de obicei cu „lățimea" (numărul de linii) ale magistralei de date. Aceasta este măsura „numărului de biți" ai unui microprocesor. Microprocesoarele cu structură fixă sînt de obicei de 8, 16 sau 32 bit. Lungimea de cuvînt a microcalculatoarelor realizate cu microprocesoarele „bit slice" (felii de bit), a căror structură este flexibilă, va fi totdeauna un multiplu întreg al numărului de bit al unei felii.

Registrul de adresă, respectiv „lățimea" magistralei de adrese definește spațiul de memorie adresabil direct de către microprocesor. O magistrală de adrese de 16 bit permite adresarea a $2^{16}=65536$ celule de memorie distincte, iar 20 de linii de adresă ne conduc în lumea megaocteților : $2^{20} = 1.048.576$ celule adresabile.

Z80 este un microprocesor pe **8** bit, cu **16** linii de adresă.

0.3. Numere binare şi reprezentarea lor hexazecimală

În interiorul calculatoarelor toate informaţiile sînt reprezentate sub forma unor numere binare, sau sub forma unor grupe de bit.

Cuvîntul **bit** derivă din **binary digit** (cifră binară). Bitul este unitatea de măsură de bază în orice sistem informaţional binar şi poate avea două valori distincte : 0 şi 1. O grupă de 8 bit se numeşte **octet** sau **byte**. O grupă de 4 bit se numeşte **nibble**.

În continuare, ne vom referi la numere binare reprezentate pe 8 bit, specificînd faptul că exemplele pe care le vom prezenta vor putea fi uşor extinse pe numere cu număr mai mare de bit.

Numerele binare se reprezintă sub forma unui şir de 0 şi 1.

Poziţiile cifrelor binare le vom numerota cu $b_7 - b_0$:

$$b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$$

b_7 reprezintă bitul (cifră binară) cel mai semnificativ iar b_0 bitul cel mai puţin semnificativ al octetului.

Valoarea unui octet se poate calcula cu expresia

$$b_7 \cdot 2^7 + b_6 \cdot 2^6 + b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Puterile lui 2 sînt :

$$2^7=128,\ 2^6=64,\ 2^5=32,\ 2^4=16,\ 2^3=8,\ 2^2=4,\ 2^1=2,\ 2^0=1$$

Aţi remarcat desigur faptul că reprezentarea binară a numerelor respectă aceleaşi reguli ca şi cea larg răspîndită, cea zecimală.

Numărul **zecimal 217** nu este altceva decît reprezentarea compactă a următorului şir de operaţii :

$$2.10^2 + 1.10^1 + 7.10^0 = 200 + 10 + 7$$

Numărul **binar 1101 1001**

este de fapt

$$2^7 + 2^6 + 2^4 + 2^3 + 1 = 217$$

În mod similar se pot defini diverse sisteme de reprezentare a numerelor, folosind ca bază de numărare orice număr mai mic sau mai mare ca 10. Pentru a ne putea familiariza uşor cu gîndul unui nou sistem de numărare trebuie să pornim de la convingerea că sistemul zecimal nu este altceva decît o cimentare profundă în creierele noastre, născută şi întărită de-a lungul mileniilor, numai şi numai datorită faptului că omul are zece degete. Dacă omul n-ar fi avut degete, ci doar două mîini, atunci mai mult ca probabil, astăzi sistemul binar ar fi cel „implementat" în sistemul nostru nervos.

Cert este că ne putem considera norocoşi, căci memorarea şi redarea (sub formă scrisă sau orală) a numerelor binare pare să fie şi un exerciţiu de răbdare. De aceea în reprezentarea externă calculatorului, numerele binare vor fi specificate de obicei într-un alt sistem de numeraţie, mai compact, dar care să se muleze bine peste organizarea **bit-nibble-byte** a numerelor binare vehiculate în calculatoare.

Sistemul cel mai potrivit s-a dovedit a fi cel **hexazecimal,** cel care admite șaisprezece cifre distincte. Cifrele propriu-zise (zecimale) se dovedesc a fi insuficiente. De aceea le vom completa cu litere. Iată deci cifrele sistemului hexazecimal : **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.**

Grupajul 10 reprezintă, ca în toate sistemele de numerație de altfel, baza sistemului hexazecimal.

$$(10_H = 16_D \; ; \; 10_B = 2_D \; ; \; 10_D = \text{„zece”})$$

Folosind cifrele hexazecimale, fiecare octet binar poate fi exprimat prin două cifre.

Aceasta este forma sub care vorbim și scriem despre numerele binare din interiorul calculatoarelor, *o punte* între structura gîndirii umane și cea a calculatoarelor.

În tab. 0.1 redăm tabela de echivalență a primelor șaptesprezece valori numerice ale sistemelor de numere zecimal, hexazecimal, binar, și octal. Ultima coloană este redată ca fapt divers :

Tabelul 0.1.

zecimal	hexa	binar	octal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17
16	10	10000	20

Asupra operațiilor aritmetice în sistemul binar,precum și asupra reprezentării numerelor binare cu semn,vom reveni în capitolul următor.

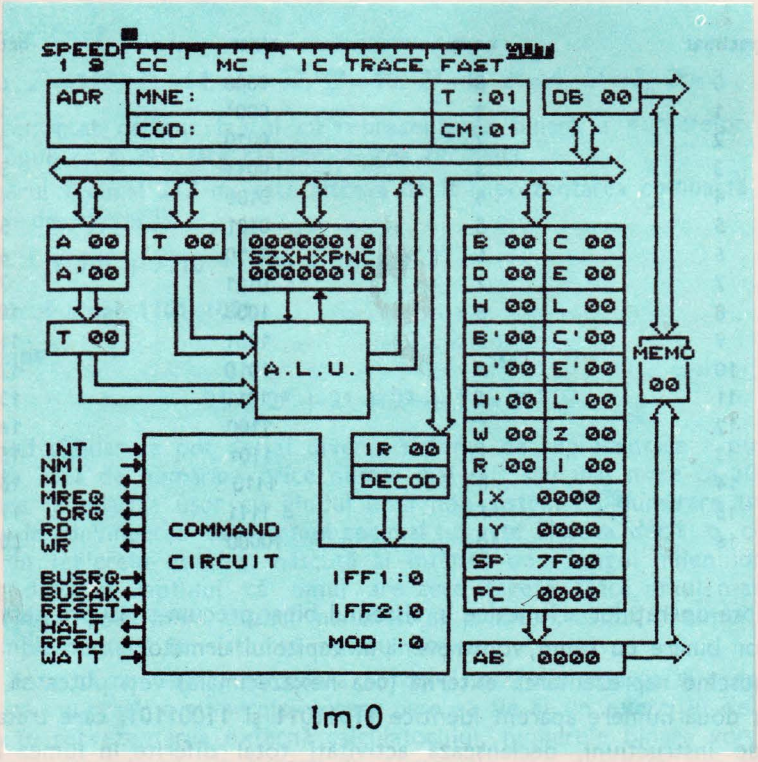
Cunoscînd reprezentarea **externă** (cea hexazecimală) veți putea să discriminați ușor două numere aparent identice **11011011** și **11001101,** care tratate drept coduri de instrucțiuni, declanșează activități total diferite în lumea internă a microprocesorului Z80. (Primul DB_H este codul instrucțiunii de citire de la periferice, iar al doilea CD_H apelează o subrutină).

1
STRUCTURA INTERNĂ

Înainte de a ne duce cu sacul la pomul lăudat, să vedem ce ne oferă la prima vedere programul **VISIBLE—Z80.** Pentru aceasta cititorul trebuie să parcurgă şi secţiunea A din partea a II-a a lucrării „COMPLEMENTE". (pag. 233)

Vom desena cu ajutorul calculatorului PRAE sau aMIC, acţionat de caseta cu programul VISIBLE—Z80, structura internă a microprocesorului **Z80,** structură organizată pe blocuri funcţionale.

Iat-o :



Im.0

În Im.0 regăsim concentrate toate structurile funcţionale pe care le-am inserat treptat în modelul microprocesorului teoretic constituit în PRAELUDIUM (vezi fig. 0.4).

În partea din stînga sus apare un cîmp de comentarii, menit să informeze utilizatorul asupra numelui şi codului instrucţiunii ce se execută, precum şi adresa de memorie la care se află ea stocată (**MNE, COD** şi **ADR**). În căsuţele notate cu **T** şi **CM** primim detalii privind faza în care a ajuns execuţia instrucţiunii studiate. Ce anume sînt ciclurile de tact şi cele maşină, vom vedea peste 50 pagini în paragraful 4.1. În extremitatea superioară a Im. 0 se observă o bară orizontală (**SPEED**) divizată în 23 de porţiuni egale, marcate din loc în loc cu simboluri (**CC, MC, IC, TRACE, FAST**). Deasupra acestei bare se poate deplasa un cursor, (o linie orizontală mai groasă) care indică pe scara gradată a barei de viteză (**SPEED**) regimul de lucru şi viteza de simulare ale programului de instruire **VISIBLE—Z80.**

Facem abstracţie de existenţa acestor elemente grafice, ele nefiind părţi componente ale microprocesorului, ci doar comentarii. Le vom folosi mai încolo, pentru a ne ghida pe drumurile accidentate ale celor zece aventuri pe care le vom întreprinde în „interiorul" microprocesorului **Z80,** în speranţa neascunsă de a descifra majoritatea tainelor acestui circuit integrat.

În extremitatea laterală dreaptă a Im. 0 apare o căsuţă notată **MEMO,** care simbolizează un bloc de memorie de lucru, fără de care nu se poate aborda prezentarea funcţionării nici unei unităţi centrale. Nici acest modul nu este o parte componentă a microprocesorului luat în studiu. Microprocesorul „se termină" de fapt la cei doi regiştri tampon : registrul tampon pentru date (**DB**-data buffer) şi registrul tampon pentru adrese (**AB** — address buffer).

Ne oprim aici cu eliminările, înainte de a-i crea cititorului impresia că dorim să reducem structura microprocesorului la un pătrăţel mic şi cenuşiu (aşa cum ea este de fapt în realitate), despre care oricum nu se poate spune mai nimic !

Rămîn deci **regiştrii speciali** şi cei **de uz general** (**AB, PC, SP, IY, IX, R, I, W, Z, H', L', D', E', B', C', H, L, D, E, B, C, A, A', F, F', T, T** şi **IR**), **circuitul de comandă** (**Command Circuit**), **unitatea aritmetică şi logică** (**A L U**) magistralele interne şi semnalele externe (**INT, NMI, M1, MREQ, IORQ, RD, WR, BUSRQ, BUSAK, RESET, HALT, RFSH** şi **WAIT**).

Să le cunoaştem pe rînd, nu înainte de a preciza faptul că structura generată de **VISIBLE—Z80,** aşa cum se vede şi în Im. 0, este o structură minimă care ţinînd cont de posibilităţile grafice limitate (256 × 256 de puncte) ale calculatoarelor personale **PRAE** şi a **MIC** încearcă să prezinte totuşi modulele de importanţă vitală, fără care prezentarea funcţionării microprocesorului ar putea duce la interpretări ambigue. Oricum, apar cîteva elemente suplimentare faţă de modelele care s-au publicat pînă în prezent în literatura de specialitate, pe care le-am considerat însă absolut necesare (de exemplu, al doilea registru T).

1.1. Regiştri

Pentru a putea funcţiona, executînd programe stocate într-un modul de memorie, precum şi pentru a avea un număr oarecare de date disponibile imediat, în interiorul fiecărui microprocesor se înglobează cîteva celule de memorare,

celule a căror conținut poate fi citit, modificat și rescris în caz de nevoie. Aceste celule de memorie de tip citește/scrie, încorporate în interiorul microprocesorului se numesc **regiștri**. Așa cum ne sugerează amplasarea și forma de reprezentare a acestor regiștri în Im. 0, ei nu sînt toți identici. Unii sînt mai scurți (8 bit), alții au lungime dublă (16 bit), la unii am reprezentat fiecare bit în parte (F și F') fiindcă ei au o semnificație aparte, la alții această discriminare nu este necesară. În Im. 0 se observă că fiecărui registru îi este atașată o căsuță în care se înscrie și valoarea hexazecimală a numărului pe care el îl conține. (Faptul că în această stare inițială conținutul majorității regiștrilor este 0, este o pură întîmplare).

Dacă dorim să clasificăm regiștrii, trei este numărul criteriilor mai des folosite :

— **lungimea** (numărul de bit) : distingem regiștri de *8 bit* (**A, B, C** etc.) și de *16 bit* (**PC, SP** etc.)

— **accesibilitatea** : pe lîngă regiștri accesibili utilizatorului prin instrucțiuni specifice, există cîțiva regiștri de manevră care se folosesc pentru stocarea temporară a unor date, regiștri care sînt *transparenți* utilizatorului (ex. **T, DB, AB** etc.)

— **destinația** : există regiștri de *uz general* care se folosesc ca simple celule de memorie internă și alții cu *destinație specială* care, pe lîngă trăsătura de memorare, fie că sînt dedicați stocării doar a unui anumit tip de date (**PC, IR, F**), fie că sînt înzestrați și cu anumite facilități suplimentare (**SP, A** etc.).

Vom începe prezentarea regiștrilor grupîndu-i după ultimul criteriu.

1.1.1. Regiștri speciali

Contorul program — PC (Program Counter) este un *registru dublu*, avînd lungimea de *16 bit*. El este destinat să memoreze adresa instrucțiunii ce urmează a fi executată. După ce se citește codul instrucțiunii curente din memorie (operația fetch), conținutul acestui registru este incrementat automat cu 1. Astfel se asigură secvențialitatea în execuția unui șir de instrucțiuni (program). Instrucțiunile se vor executa începînd de la adresa 0, în ordinea crescătoare a adreselor. Există și instrucțiuni de salt, care înscriu direct conținutul registrului PC cu o valoare dorită de programator. Astfel pot fi executate programe a căror adresă de început diferă de 0 și se pot executa ramificații în program, în funcție de rezultatul unor calcule, sau a unor evenimente externe detectate prin program (salturi condiționate). Memoria adresabilă direct este de $2^{16} = 65536$ locații de memorie distincte. Spațiul de adrese este deci 0000—FFFF în sistem hexazecimal.

Indicatorul de stivă — SP (Stack Pointer) este un *registru dublu*, avînd lungimea de *16 bit*. **SP** este un registru de adresare special. Avînd aceeași lungime ca și PC, el poate adresa întregul spațiu de adrese de memorie : 0000—FFFF Pentru a înțelege mai bine menirea lui, trebuie să facem o mică paranteză.

Stim că numărul regiștrilor interni ai microprocesorului este limitat și, implicit, și cel al datelor ce se pot înmagazina în ei. În cursul elaborării unor programe scrise în limbaj de asamblare, apare frecvent necesitatea de a elibera unul sau mai mulți regiștri pentru ca în ei să se poată înmagazina temporar noi date. Nedorind să pierdem informația precedentă din regiștrii respectivi, îi vom salva în

memoria externă **RAM** (Random Access Memory), care fiind o memorie de tip scrie/citeşte permite să restaurăm în regiştri oricînd valorile eliminate temporar. Se pune întrebarea : unde anume să salvăm aceste date, la ce adresă ? Dificultatea derivă din două surse : pe de o parte, dacă am dori să efectuăm aceste salvări la adrese de memorie nominalizate, s-ar putea să fim nevoiţi să memorăm la un moment dat zeci de adrese de memorie, şi semnificaţia fiecăreia (ce conţine celula respectivă) fapt care ar îngreuna considerabil munca de programare. Pe de altă parte, se ştie că de la John von Neumann încoace în memoria RAM poate rezida şi programul executat. Dacă se execută salvări repetate ale regiştrilor se poate ivi la un moment dat un conflict : valorile salvate riscă să se suprapună peste program, lezîndu-l, ceea ce trebuie să recunoaştem, ar fi o eroare fatală. Pentru a se depăşi inconvenientele prezentate, s-a convenit ca anumite zone de memorie să aibă o organizare specială, reglementîndu-se acceptul la ele (adresarea) pe baza unor reguli universal valabile şi simple. Se va folosi un indicator de adresă care la fiecare salvare de registru (scriere în memorie) va avansa automat, indicînd adresa următoarei celule libere. La restaurarea conţinutului registrului dorit (citire din memorie) indicatorul de adresă se va repoziţiona automat, indicînd adresa de memorie a penultimei date salvate. O zonă de memorie astfel organizată se numeşte stivă. Legea de bază care guvernează mişcarea de date în această zonă se numeşte : **LIFO** (Last In First Out), adică ultimul intrat este primul care pleacă. Legea este simplă şi o cunoaştem cu toţii.

Observăm că dacă există un registru de adresă prevăzut cu automatul de incrementare/decrementare la operaţiile de scriere şi citire memorie, atunci el poate controla în mod univoc toate activităţile de salvare/restaurare în stivă. Rezultă deci, că stiva poate fi organizată oriunde într-o memorie de tip scrie/citeşte (RAM). Registrul de adresare al stivei se numeşte indicator de stivă. La majoritatea microprocesoarelor stiva este descrescătoare : salvările (scrierile) succesive se fac la adrese descrescătoare. S-a ales această variantă pornind de la faptul că contorul program **PC** variază crescător. Programele locate în memorie încep de la adresa CCC0 şi cresc spre adrese superioare. Pentru a nu se disloca mereu zona de stivă, în măsura în care lungimea programelor variază, este convenabil ca indicatorul de stivă să fie iniţializat astfel încît să ponteze (adreseze) sfîrşitul memoriei RAM, de unde va descreşte la salvări. O altă soluţie de separare a zonei program de zona de stivă este aceea de a amplasa stiva la adresa imediat inferioară începutului de program. Şi în acest caz avem nevoie de o stivă descrescătoare.

La microprocesorul **Z80**, registrul **SP** organizează o *stivă descrescătoare* : la fiecare **salvare** se înscriu în memorie 2 octeţi (2 regiştri simpli sau 1 registru dublu) şi conţinutul indicatorului de stivă este **decrementat** cu 2.

$$SP = SP - 2$$

La fiecare **restaurare** se citesc din memorie 2 octeţi şi conţinutul indicatorului de stivă este **incrementat** cu 2.

$$SP = SP + 2$$

Pentru salvarea şi restaurarea datelor din regiştri, stiva se dovedeşte a fi foarte utilă şi convenabilă, dar nu şi de neînlocuit. Există în schimb un caz în care existenţa unei memorii organizată ca stivă este imperios necesară, şi anume : la apelarea subrutinelor şi mai ales în cazul unor apeluri multiple.

Ori de cîte ori programatorul scrie anumite **module** care pot fi folosite în mai multe puncte ale programului său, el le va scrie sub formă de subrutine. O subrutină poate fi oricînd apelată dintr-un program principal, folosind o instrucţiune de tipul **CALL adresă**, unde **adresă** este adresa de început a subrutinei. La întîlnirea unei instrucţiuni **CALL** secvenţa de execuţie a programului principal este abandonată, se execută subrutina, iar la întîlnirea unei instrucţiuni de tipul **RET**, care marchează sfîrşitul subrutinei, se revine în programul apelant, la instrucţiunea imediat următoare instrucţiunii **CALL**. Cum se revine ?

Odată cu executarea instrucţiunii **CALL**, adresa instrucţiunii următoare din programul apelant este salvată în vîrful stivei, iar la execuţia instrucţiunii **RET** această valoare este citită din vîrful stivei şi încărcată în contorul program **PC**, reluîndu-se astfel „firul pierdut". La citirea acestei pledoarii pentru necesitatea imperioasă a indicatorului de stivă **SP**, oricine şi-ar putea pune întrebarea : nu s-ar fi putut folosi un registru tampon, dedicat pentru salvarea adresei de revenire în programul apelant ? S-ar fi putut, da ! Dar ce ne facem, dacă dintr-o subrutină dorim să apelăm o altă subrutină, şi aşa mai departe ? Pentru fiecare nivel de întreţesere în apelarea subrutinelor ar fi necesar cîte un registru suplimentar pentru memorarea tuturor adreselor de revenire. În istoria microprocesoarelor se cunoaşte şi această soluţie. Primul microprocesor pe 8 bit, microprocesorul **I8008** al firmei INTEL, avea 8 regiştri interni pentru memorarea adreselor de revenire. S-a şi renunţat la această idee, deja la următorul microprocesor, **I8080** al aceleiaşi firme. Soluţia iniţială s-a dovedit a fi neavantajoasă, fiindcă limita nivelul de întreţesere a subrutinelor la 8 şi consuma totodată o cantitate considerabilă din resursele hardware interne ale microprocesorului. Iată de ce un indicator de stivă care să permită organizarea unei zone de stivă în memoria RAM, externă microprocesorului, este într-adevăr indispensabil.

Atragem de pe acuma atenţia cititorului că folosirea acestui registru este delicată. Pe stivă se înmagazinează atît valori salvate din regiştri de date, cît şi adrese de revenire din subrutine. Vă daţi seama ce se întîmplă dacă dintr-o eroare de programare, la revenirea dintr-o subrutină în contorul program **PC** se încarcă o dată oarecare şi nu adresa de revenire ! Pentru eliminarea acestui inconvenient la microprocesoarele mai evoluate există doi sau mai mulţi indicatori de stivă, putîndu-se organiza astfel stive separate pentru date şi pentru adrese de revenire din subrutine. Pînă una alta, specialiştii firmei Zilog au păstrat un singur indicator de stivă pentru microprocesorul **Z80**. Aşa vom face şi noi.

Registrul de instrucţiune — IR (Instruction Register) este un *registru special* cu lungimea de *8 bit*. Aceasta este celula în care se depozitează codul unei instrucţiuni, citit din memorie la operaţia fetch. Circuitul de comandă al microprocesosului preia codul instrucţiunii de executat din **IR**, îl decodifică şi îşi programează toate activităţile pe care le are de efectuat în vederea ducerii la bun sfîrşit a „poruncii" primite. Programatorul nu are acces în mod explicit la acest registru.

Registrul A (Acumulator) — este un registru avînd lungimea de 8 *bit*. El se distinge prin faptul că pe lîngă a şti să memoreze un octet, este implicat în toate operaţiile aritmetice şi logice : unul din cei doi parametri asupra cărora urmează să fie efectuată o operaţie aritmetică sau logică (adunare, scădere, etc.) se va afla *obligatoriu* în acumulator. Rezultatul operaţiei se generează de asemenea în registrul A, el suprascriindu-se peste valoarea iniţială. Observăm că operaţiile

aritmetice şi cele logice acţionează astfel încît acumulează rezultatele în registrul **A**. De aici provine şi numele de *acumulator*. Ne putem imagina un microprocesor virtual care să aibă mai mulţi regiştri prevăzuţi cu această facilitate specială. Acel microprocesor ar avea mai mulţi acumulatori. S-au realizat şi astfel de microprocesoare. Un exemplu în acest sens îl poate constitui microprocesorul **M6800** al firmei **Motorola**. Conchidem prin a afirma că a poseda mai mulţi acumulatori poate fi un avantaj, reducîndu-se astfel numărul de manevre necesare pentru efectuarea unor operaţii aritmetice/logice, dar este cert : un singur acumulator este suficient pentru a efectua operaţii oricît de complexe. Totul depinde de calificarea profesională şi iscusinţa programatorului în a aranja operaţiile elementare ale microprocesorului într-o ordine dorită, astfel ca programul realizat să efectueze într-adevăr ceea ce dorim.

În Im. 0 remarcăm prezenţa unei căsuţe marcate cu simbolul **A'**. Acest registru este un *registru secundar*, care împreună cu regiştrii **F'**, **B'**, **C'**, **D'**, **E'**, **H'** şi **L'** formează grupul regiştrilor secundari. Existenţa acestor regiştri secundari este o particularitate a microprocesorului **Z80**. Ei sporesc considerabil performanţele lui **Z80** în comparaţie cu predecesorul său **I 8080**. Conţinutul registrului **A'** poate fi interschimbat cu cel al registrului **A**, folosind instrucţiunea **EX AF, AF'**. **A'** nu este un al doilea acumulator al microprocesorului **Z80** fiindcă doar unul, anume **A**, poate fi *activ* la un moment dat. *Toate instrucţiunile microprocesorului afectează conţinutul acumulatorului primar* **A**.

Registrul indicatorilor de condiţie — F (Flag). Aşa cum o spune şi numele simbolic ales pentru acest *registru special* (flag-fanion, steguleţ) el este menit să semnaleze ceva. În acest registru, care în cazul microprocesorului Z80 are lungimea de *8 bit*, fiecare bit în parte are semnificaţia lui. Starea lor (0 sau 1) indică satisfacerea sau nesatisfacerea unor condiţii date. De aceea ei se numesc indicatori de condiţie. Flagurile furnizează atît programatorului cît şi însăşi unităţii de comandă a microprocesorului, informaţii privind natura rezultatului unei operaţii aritmetice sau logice efectuate. Din starea *biţilor dedicaţi* ai registrului **F** putem afla dacă rezultatul operaţiei efectuate este 0 sau nu, dacă el este un număr pozitiv sau negativ, dacă numărul biţilor din rezultat ce au valoarea 1 este un număr par sau impar etc.

Să-i privim pe rînd :

● *Flagul* **S** (*sign=semn*) care apare pe poziţia cea mai semnificativă a registrului **F** (**bit 7**), memorează bitul cel mai semnificativ al unui număr rezultat pe baza unei operaţii aritmetice sau logice, în acumulator. În reprezentarea binară a numerelor, bitul cel mai semnificativ se foloseşte adesea pentru indicarea semnului unui număr. Prin *convenţie*, şi rezultînd din particularităţile reprezentării sub forma complementului faţă de 2 a numerelor binare, valoarea **0** a bitului cel mai semnificativ indică **număr pozitiv**, iar valoarea **1** un **număr negativ**. Înzestrînd bitul cel mai semnificativ al unui număr binar avînd lungimea de *n* bit, cu funcţia de semn, totalul numerelor reprezentabile prin acel număr nu se schimbă. Se modifică însă plaja de reprezentare a lor. Dacă se foloseşte convenţia *numerelor întregi fără semn*, atunci un număr binar de 8 bit poate avea $2^8 = 256$ de valori distincte, cuprinse în plaja [0,255]. Dacă convenţia de reprezentare este : *numere întregi cu semn*, atunci aceeiaşi 8 bit pot reprezenta deasemenea 256 de valori, cuprinse de data aceasta în plaja [−128, +127]. În tehnica de programare a microprocesoarelor, rămîne pe seama programatorului să decidă asupra semni-

ficației bitului celui mai semnificativ și implicit al flagului de semn **S**. Starea flagului **S** poate fi testată de către programator prin două instrucțiuni de salt condiționat : **JP P,** nn și **JP M,** nn care permit ramificarea programului în funcție de starea indicatorului de semn.

JP P,nn (jump if plus) — salt dacă pozitiv, efectuează saltul la adresa **nn**, dacă **S**=0 ;

JP M,nn (jump if minus) — salt dacă negativ, efectuează saltul la adresa **nn**, dacă **S**=1.

Vom vedea mai încolo că există și alte instrucțiuni prin care se poate testa flagul **S**, ca și flagurile **Z**,**P/V**, și **Cy** de altfel, dar ne limităm deocamdată la aceste două, fiindcă ele ilustrează destul de sugestiv utilitatea indicatorilor de condiție.

● *Flagul* **Z** (*zero*) — apare pe poziția **bit 6** a registrului **F**. El este înscris (valoarea 1), dacă rezultatul unei operații aritmetice/logice este 0. Dacă rezultatul operației diferă de 0, atunci conținutul indicatorului Z va fi 0. Ca și **S**, flagul **Z** poate fi testat de către programator prin instrucțiunile de salt condiționat : **JP NZ,**nn ; **JP Z,**nn unde :

JP NZ,nn (jump if not zero) semnifică salt, dacă diferit de zero : saltul se efectuează dacă **Z**=0 ;

JP Z,nn (jump if zero) înseamnă salt, dacă zero : saltul se efectuează dacă **Z** = 1. Acest indicator este folosit și de către circuitul de comandă al microprocesorului, la execuția unor instrucțiuni repetitive. Amintim instrucțiunile de comparație **CPDR** și **CPIR** (compare increment repeat respectiv decrement...) caz în care instrucțiunea care compară conținutul acumulatorului cu celule de memorie aflate la adrese crescătoare (sau descrescătoare) se termină dacă **Z**=1, adică în cazul egalității celor două numere comparate. Instrucțiunile de transfer de blocuri de date la periferice, **INDR** și **INIR** precum și **OTDR** și **OTIR** se termină deasemenea în cazul în care **Z**=1 (conținutul registrului B devine 0).

● *Flagul* **H** (*Half carry*) reprezintă *transportul* care apare în cursul unei operații aritmetice de la *bit 3* spre *bit 4* al acumulatorului. El ocupă poziția **bit 4** în registrul indicatorilor de condiție **F**. Flagul **H** nu poate fi testat prin instrucțiuni de salt condiționat. El este folosit de către unitatea aritmetică/logică a microprocesorului **(ALU)**, la execuția instrucțiunii de corecție zecimală **(DAA** — Decimal Adjust Acumulator), pentru corecția numerelor zecimale codificate binar : BCD. Dacă considerăm cei 4 biți inferiori (bit 0 — bit 3) ai unui număr ca o cifră hexazecimală (digit), care poate avea valori în domeniul [0 — F], transportul intermediar **H** va fi închis (va avea valoarea 1) ori de cîte ori, ca rezultat a unei operații de adunare sau scădere, valoarea digitului mai puțin semnificativ (bit 0 — bit 3) depășește domeniul : fie că ar trebui să fie mai mare de 9, fie că ar trebui să fie negativ (mai mic ca zero). ALU execută toate operațiile în sistemul binar. De aceea, pentru a putea lucra și cu numere zecimale codificate în binar (BCD), instrucțiunea de ajustare a rezultatului unei operații aritmetice **DAA**, va folosi acest bit de transport intermediar. *

● *Flagul* **P/V** (*Parity/Overflow*) — este un indicator de condiție *multifuncțional*. El ocupă poziția **bit 2** în registrul **F**. Așa cum rezultă din numele acestui indicator, el poate indica *paritatea* numărului din acumulator sau *depășire de domeniu*. O a treia funcție a acestui flag este citată destul de rar în literatura de specialitate (ea dezvăluindu-se doar celor care studiază atent setul de instrucțiuni al microprocesorului) : instrucțiunile de transfer multiple **(LDD, LDI, LDDR, LDIR)** și cele logice multiple **(CPD, CPI, CPDR, CPIR)** utilizează flagul **P/V** ca *detector de zero al registrului dublu* **BC**. În acest din urmă caz **P/V**=0, dacă

* Notat în Im. 0 cu P.

conținutul registrului dublu **BC**=0, ceea ce este o condiție de terminare a instrucțiunilor cu repetiție. Pentru aceste instrucțiuni **P/V**=1 dacă **BC**≠0.
Dar să revenim la funcțiile de bază ale indicatorului **P/V**.

Paritatea

Una din procedeele de verificare a corectitudinii transferurilor de date intra-și intercalculator este cea care atașează celor *n* biți utili de date un bit suplimentar de control. Acest bit se alege astfel încît suma cifrelor 1 din totalul celor *n*+1 biți transferați să satisfacă criteriul de paritate ales : **paritate pară (even)** — suma cifrelor de 1 este un număr par ; **paritate impară (odd)** — suma cifrelor de 1 este un număr impar. Nu încape îndoială că oricine ar putea concepe un program scurt care să calculeze paritatea. Sîntem însă scutiți de acest efort, datorită existenței flagului **P/V** care în anumite condiții (vom vedea care) indică tocmai paritatea numărului binar din acumulator. Atunci cînd **P/V** indică paritatea el are următoarele valori :

P/V=0 dacă totalul cifrelor 1 din **A** este un număr **impar** ;
P/V=1 dacă totalul cifrelor 1 din **A** este un număr **par**.

Depășirea

Pentru a înțelege mai bine noțiunea de depășire și rolul indicatorului **P/V** în mod de depășire, trebuie să ne întoarcem la reprezentarea *numerelor binare cu semn* : bitul cel mai semnificativ, bit 7 (în cazul microprocesoarelor pe 8 biți) este bit de semn. Folosind pentru numere negative reprezentarea în complementul față de 2, avem :

0000 0000 — 00_H — 0

0111 1111 — $7F_H$ — +127

1000 0000 — 80_H — −128

1111 1111 — FF_H — −1

Numim depășire fenomenul prin care rezultatul unei operații aritmetice depășește domeniul [−128, +127].
Dar de unde aflăm, dacă avînd un rezultat de

1000 1111

el reprezintă : a) numărul **−113** (încadrat în domeniu)
sau b) numărul **+143=128+15** (depășind domeniul de valori) ?
Pentru a putea fi siguri de corectitudinea rezultatului, ar trebui să analizăm cele 2 numere implicate în operația aritmetică efectuată, și astfel am putea prezice dacă se va depăși domeniul de valori sau nu.
Regimul de „depășire" al indicatorului **P/V** ne scutește de această antecalculație. Dar cum ? Vom încerca să stabilim o procedură, ușor implementabilă, pentru a vedea modul de funcționare a flagului **P/V**.

Exemplul 1 :

bit :	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**adică :**
nr. 1 :	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**+95+**
nr. 2 :	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**+99**
rez. :	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**+194**
transp. :	**1**	**1**	**1**	**1**	**1**	**1**	**1**		

Comparînd rezultatul adunării binare (11000010=—62) cu cel al operației zecimale (+194) observăm că rezultatul binar diferă de cel zecimal, fiind deci incorect. Analizînd faptul că am adunat două numere pozitive +95 și +99 și am obținut un număr negativ —62, eroarea o atribuim depășirii domeniului de valori. *Eroarea se datorește existenței unui transport de la bit 6 la bit 7 (bitul de semn).* Această depășire trebuie detectată, pentru ca rezultatul să poată fi corectat.

Exemplul 2 :

bit : 7 6 5 4 3 2 1 0		adică :
nr. 1 : 1 1 1 1 1 1 0 0	+	—4+
nr. 2 : 1 1 1 1 1 1 1 0		—2
rez. : 1 1 1 1 1 1		—6
transp. : 1 1 1 1 1 1		

adică : —6 — corect.

Observăm că și în acest caz apare un *transport* de la *bit 6* la *bit 7*, dar acesta nu afectează corectitudinea rezultatului, deoarece atît cele două numere adunate, cît și cel rezultat au semnul negativ. Observăm în schimb, că apare spre deosebire de primul caz și un *transport* de la *bit 7* în sus (acest transport se numește pe obicei *carry*).

Exemplul 3 :

bit : 7 6 5 4 3 2 1 0		adică :
nr. 1 : 1 1 0 1 0 1 1 0	+	—42+
nr. 2 : 1 0 0 1 1 1 1 1		—97
rez. : 0 1 1 1 0 1 0 1		—139
transp. : 1 1 1 1		

adică : +117 — incorect.

În acest caz numărul rezultat din adunarea binară a două numere negative este un număr pozitiv, ceea ce desigur este incorect. În acest caz *nu a existat transport de la bit 6 la bit 7, dar a apărut un transport de la bit 7 în sus (carry).*

Exemplul 4 :

bit : 7 6 5 4 3 2 1 0		adică :
nr. 1 : 1 0 1 0 1 1 1 1	+	—81+
nr. 2 : 0 1 1 0 0 1 1 0		+102
rez. : 0 0 0 1 0 1 0 1		+21
transp. : 1 1 1 1 1 1		

adică : +21 — corect. 0

Observăm că și în acest caz rezultatul adunării binare este corect. *A existat transport atît de la bit 6 spre bit 7 cît și de la bit 7 în sus.*
Din cele patru exemple putem să tragem concluzia că rezultatul operației aritmetice este corect, dacă apare concomitent transport către bitul de semn și de la el în sus. Dacă apare numai unul din aceste transporturi, atunci rezultatul va trebui să fie ajustat, căci a apărut condiția de depășire a domeniului de valori. *Această condiție se poate implementa ușor prin hardware, realizînd o funcție SAU EXCLUSIV între biții de transport către și de la bitul de semn :*

$$V = T_6 \oplus T_7$$

Concluzie : după operații aritmetice (ADD, ADC, SUB, SBC, INC, DEC) flagul P/V indică apariția unei depășiri de domeniu :

P/V = 1 a existat depășire ;

P/V = 0 nu a existat depășire.

Dacă numerele cu care lucrăm sînt reprezentate cu semn și dacă apare depășire rezultatul trebuie corectat.

Flagul P/V poate fi testat prin instrucțiuni de salt condiționat : **JP PO, nn** și **JP PE, nn.**

JP PO, nn (jump if parity odd) se execută salt la adresa **nn,** dacă **P/V=0 ;**

JP PE, nn (jump if parity even) se execută salt la adresa **nn,** dacă **P/V=1.**

Remarcăm, că indiferent de faptul că P/V indică paritate sau depășire, mnemonica face referire mereu la paritate. Aceasta este o *posibilă capcană pentru programatori*, ei trebuind să știe ce anume indică flagul P/V în momentul în care se folosește una din instrucțiunile de salt condiționat ce testează starea acestui bit.

Este momentul să facem o observație importantă.

Acesta este punctul în care mult amintita compatibilitate a microprocesorului Z80 cu predecesorul său I8080 șchioapătă un pic : dacă dorim ca toate programele scrise pentru microprocesorul I8080 să funcționeze ireproșabil și pe Z80 atunci va trebui să verificăm toate instrucțiunile care testează flagul P/V — JP PO ; JP PE ; CALL PO ; CALL PE ; RET PO ; RET PE. Ele vor cauza probabil salturi eronate pe Z80, dacă inițial (pe I8080) au fost folosite pentru testarea parității după efectuarea unor instrucțiuni aritmetice. Pe Z80, P/V va indica în acest caz depășirea și nu paritatea. Spre norocul softiștilor experiența arată că aceste instrucțiuni de salt sînt mai rar folosite, și chiar în cazul utilizării lor majoritatea programatorilor le-au folosit după instrucțiuni logice. Oricum, atenție !

● *Flagul* **N** — este un indicator care memorează tipul ultimei operații aritmetice efectuate : adunare sau scădere. În registrul **F** el ocupă poziția **bit 1.** De ce se numește tocmai **N,** nu știm nici noi, referințele bibliografice avute la dispoziție neindicînd nimic sugestiv în acest sens.

N=1 dacă ultima operație aritmetică a fost **adunare ;**

N=0 dacă ultima operație aritmetică a fost **scădere.**

Flagul **N** este folosit împreună cu indicatorul de transport intermediar **H** și cu indicatorul de transport **Cy,** pentru ajustarea numerelor zecimale codificate binar (BCD), de către instrucțiunea **DAA.**

Utilizatorul nu dispune de instrucțiuni dedicate pentru testarea valorii acestui indicator. *

● *Flagul* **Cy** *(Carry — transport)* — este indicatorul cel mai popular, afectat atît de operațiile aritmetice/logice cît și de clasa operațiilor de rotire/deplasare octet. El ocupă poziția cea mai puțin semnificativă **(bit 0)** în registrul indicatorilor de condiție **F.**

Flagul **Cy** este înscris (valoarea 1), ori de cîte ori apare un transport de la cifra cea mai semnificativă în sus. Acesta este cazul, dacă pe urma unei adunări în aritmetica fără semn (gama de valori posibile 0—255) rezultă un număr mai mare de 255. La scădere **Cy** = 1, dacă scăzătorul este mai mare decît descăzutul.

Spre deosebire de toți ceilalți indicatori de condiție, în cazul flagului **Cy** există instrucțiuni dedicate pentru modificarea conținutului acestui bit :

SCF (Set Carry Flag) cauzează **Cy=1,** iar

CCF (Complement Carry Flag) complementează conținutul **Cy,** Existența acestor instrucțiuni ne sugerează ideea să folosim indicatorul **Cy** ca indicator universal de condiție. Un programator își poate fixa de exemplu regula ca **Cy** să fie

* Notat în Im. **0** prin C.

folosit ca indicator de eroare : dacă la întoarcerea dintr-o subrutină **Cy=1**, ieşirea din subrutină să fie anormală. În acest caz programul apelant va abandona calea de prelucrare uzuală a rezultatelor subrutinei, şi va trece la analiza şi tratarea erorilor. Dacă **Cy=0**, ieşirea din subrutină să fie normală, programatorul putînd prelucra datele primite conform procedurii „interesante". Dar acest exemplu este doar o idee, oricine putînd ataşa alte semnificaţii posibile bitului **Cy**.
Indicatorul **Cy** poate fi testat prin instrucţiuni de salt condiţionat : **JP NC,** nn şi **JP C,** nn.
Folosind aceste instrucţiuni ramificaţia programului se va face după cum urmează :

JP NC, nn (jump if non carry) — salt la **nn,** dacă **Cy=0**
JP C, nn (jump if carry) — salt la **nn,** dacă **Cy=1.**

Ca şi în cazul celorlalte flaguri testabile prin instrucţiuni de salt condiţionat şi **Cy** poate dirija apeluri şi reveniri condiţionate din subrutine :

CALL NC, nn ; **CALL C,** nn ; **RET NC** ; **RET C.**

Odată cu prezentarea indicatorului de transport **Cy,** am terminat prezentarea registrului **F.** Aţi remarcat poate o mică neconcordanţă : iniţial am afirmat că registrul **F** are lungimea de **8** bit, iar pe parcurs n-am prezentat decît **6.**
Doi biţi ai registrului **F** *sînt lipsiţi de semnificaţie, conţinutul lor variind imprevizibil pe toată durata de funcţionare a microprocesorului Z80* (notaţi cu X în Im. 0).
Structura registrului **F** va fi deci :

F:	S	Z	X	H	X	P/V	N	Cy

Fig. 1.1. Structura registrului F

În Im. 0, **VISIBLE—Z80** redă în mod asemănător structura registrului **F.** Remarcăm prezenţa a două numere binare de 8 biţi în căsuţa afectată registrului **F.** Ca şi în cazul acumulatorului **(A şi A'),** există un registru **F'** numit **registru de flaguri secundar.** Dar şi în acest caz doar unul, anume **F,** este activ. Conţinutul celor doi regiştri indicatori de stare poate fi interschimbat prin instrucţiunea **EX AF, AF'** . *Remarcăm faptul că toate operaţiile aritmetice/logice, care afectează indicatorii de condiţie, precum şi instrucţiunile care le folosesc, acţionează totdeauna asupra valorilor cuprinse în registrul primar* **F.**

Registrul de reîmprospătare a memoriilor dinamice R (Refresh) — este un registru cu lungimea de **7 bit,** menit să asigure printr-o numărare ciclică de **la 0** la **127 (0000000—1111111)** reîmprospătarea **memoriilor RAM dinamice,** din componenţa unui microcalculator, atunci cînd ele există.
Pentru a-i înţelge mai bine menirea şi modul în care se achită de sarcina-i încredinţată, vom face din nou o paranteză.

În memoriile RAM dinamice păstrarea informaţiei se face în celule a căror element de memorare este un condensator realizat în structura semiconductorului. Ştiind că un circuit de memorie dinamică avînd 65536 celule de memorie distincte este un circuit integrat cu 16 pini, a cărui structură (chip) nu depăşeşte dimensiunea de 3 × 3 mm, ne putem da seama că şi dimensiunile geometrice ale condensatoarelor de memorare şi implicit capacitatea lor este foarte mică. Capacitatea unui condensator de memorare într-un circuit RAM dinamic nu depăşeşte 10^{-15}F. Datorită curenţilor de scurgere prin dielectricul acestor condensatoare, ele au tendinţa de a se descărca, pierzînd astfel în timp treptat informaţia (1 sau 0) înregistrată în ele. Timpul limită în care această descărcare nu periclitează încă integritatea informaţiei stocate, este de aproximativ 2 ms. *De aceea din 2 în 2 ms condensatoarele vor trebui repolarizate la valoarea lor iniţială.* Această procedură se numeşte *reîmprospătarea memoriei dinamice (refresh).* Care este

tehnica folosită ? Pentru a da un răspuns la această întrebare vom considera structura circuitelor DRAM.

La ora răspîndirii pe scară largă a microprocesoarelor de 8 bit cele mai răspîndite circuite DRAM au fost cele de 16 kbit. Ele au o organizare matricială avînd 128 linii și tot atîtea coloane. Astfel se acoperă întregul spațiu de adrese de 16 kbit : $128 \times 128 = 16.384 = 16$ kbit. Memoriile dinamice sînt astfel realizate, încît selectarea unei celule pentru scriere sau citire reîmprospătează întregul rînd pe care se află elementul accedat. Ba chiar mai mult, nici nu trebuie făcut accesul pînă la celula dată ; este suficient să se selecteze rîndul respectiv. *Reîmprospătarea memoriei dinamice se poate face în acest caz asigurînd ca toate cele 128 de rînduri să fie selectate cel puțin o dată la 2 ms.* Cele **128** de rînduri se pot codifica pe **7** linii **(2^7=128)** de adresă. De aici rezultă că un **registru numărător circular** avînd lungimea **de 7 bit** ar putea sta la baza sistemului de reîmprospătare. (Același număr de biți — 7 — este suficient și în cazul utilizării unui circuit RAM dinamic avînd capacitatea de 64 kbit, căci aceste circuite conțin de fapt 4 grupe de 16 kbit). *Iată de ce registrul* **R** *are lungimea de 7 bit.* Rămîne de văzut cum funcționează. Proiectanții microprocesorului **Z80** au plecat de la ideea că în timpul decodificării unei instrucțiuni citite din memorie, microprocesorul nu efectuează nici o activitate externă. Magistrala de adrese a microprocesorului își poate schimba deci conținutul, pe biții ei inferiori **A0** — **A6** apărînd conținutul registrului **R**, care acompaniat de 2 semnale de comandă (**$\overline{\text{MREQ}}$** și **$\overline{\text{RFSH}}$**) va putea asigura reîmprospătarea memoriei dinamice.

După cum ați realizat probabil, registrul **R** este înzestrat și cu un mecanism care îi incrementează automat conținutul la fiecare ciclu refresh. (Dacă microprocesorul **Z80** ar executa numai instrucțiuni cu un singur ciclu mașină, ex. **NOP**, conținutul primilor 7 biți ai contorului program **PC** și cei ai registrului **R** ar varia sincron. Vizualizînd liniile de adrese pe un osciloscop, adresele **A0** — **A6** ar fi „curate", pe cînd **A7** — **A15** ar avea suprapuse peste porțiunea „1" variația de frecvență egală cu cea a lui **A0**, datorată desincronizării celor două numărătoare (**PC** și **R**), desincronizare care apare de la **A7** în sus.

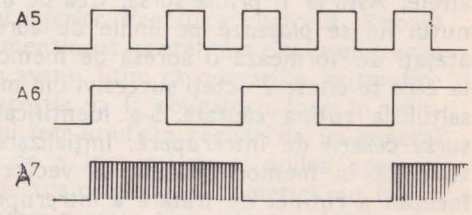


Fig. 1.2. Variația liniilor de adresă în cazul execuției permanente a unei instrucțiuni cu un singur ciclu mașină de 4 tacți (ex. NOP)

Conținutul registrului **R** poate fi înscris și citit „via" acumulator :

LD R,A — *înscrie* **R**

LD A,R — *citește* **R**

Acestea sînt singurele instrucțiuni prin care programatorul poate accede la registrul **R**. În rest el își trăiește viața monotonă și autonomă de numărător circular de 7 bit.

La inițializarea microprocesorului (aplicarea semnalului de comandă **$\overline{\text{RESET}}$**), conținutul registrului de refresh **R** este încărcat cu 0.

Prezența registrului **R** *este un cîștig semnificativ în favoarea microprocesorului* **Z80**, *ea scutindu-l pe proiectantul microcalculatorului de un efort considerabil, cel de a realiza o memorie cu 2 căi de acces (din partea procesorului și din partea*

unui circuit de reîmprospătare extern) și de a rezolva conflictele ce apar între cele două subansamble.

Registrul vectorului de întreruperi I (Interrupt register) — este ultimul în șirul regiștrilor *speciali* ai microprocesorului **Z80.** El este un registru cu lungimea de *8 bit* și servește în *modul de întreruperi 2* **(IM 2)** la dirijarea (identificarea) sursei de cerere a întreruperilor.

Prin întrerupere înțelegem fenomenul la apariția căruia microprocesorul abandonează — la cererea unui eveniment extern — programul în curs de rulare, deservește — executînd un program dedicat — evenimentul extern, după care se reîntoarce la programul abandonat, reluîndu-i execuția din punctul în care ea fusese suspendată.

Sînt multe aspecte care trebuiesc lămurite în legătură cu acest fenomen. Asupra lor vom reveni într-una din aventurile realizate cu ajutorul lui **VISIBLE—Z80.** Ne oprim acum doar asupra unui aspect : *cum identifică microprocesorul perifericul (sau evenimentul extern) care a solicitat întreruperea ?* Cererea de întrerupere **INT** sosește fizic la microprocesor pe linia de stare avînd același nume. Pentru identificarea apelantului, și implicit pentru localizarea rutinei de tratare aferente s-au elaborat numeroase soluții, și se pot imagina cel puțin atîtea noi variante. Soluția cea mai rapidă și cea mai flexibilă pare a fi **identificarea vectorizată** : *din diverse surse se constituie o adresă de memorie, care este însăși adresa de început a rutinei de tratare a cererii de întrerupere recepționate.* Această adresă se încarcă în contorul program **PC** și microprocesorul a și ajuns la rutina dorită. Care sînt aceste surse ? În cazul microprocesorului **Z80,** *perifericul* care solicită întreruperea *va plasa,* o dată cu acceptarea cererii ei, pe liniile de date *un octet de adresă,* pentru identificare. Fiecare periferic are adresa lui proprie, inconfundabilă cu altele. Asta ar fi prima sursă. Cea de a doua sursă este însăși registrul **I.** Conținutul lui *se plasează* pe liniile de adresă superioare **A8—A15.** Din cei 2 octeți atașați se formează o adresă de memorie **A0—A15** *(vector de întrerupere)* de la care se citesc 2 octeți succesivi din memorie, și se încarcă în **PC** efectuîndu-se saltul la rutina căutată. S-a identificat astfel implicit, fără căutări sau testări, sursa cererii de întrerupere. Inițializarea registrului **I,** precum și asigurarea faptului ca în memorie, la adresa vectorului de întrerupere, să se afle adresa de început a rutinei de tratare a întreruperii respective, cade bineînțeles în sarcina programatorului. *Procedura de adresare* folosită este *dublu indirectă. A doua* treaptă de adresare — cea de citire a adresei efective de început a subrutinei de tratare a întreruperii acceptate,— *este indirectă, prin memorie.* Începătorii vor putea găsi detalii privind tehnicile de adresare în paragraful : **„Tehnici de adresare".** Programatorul accede registrul **I,** ca și registrul **R,** „via" acumulator :

LD I,A — *înscrie* o valoare în **I** ;

LD A,I — *citește* valoarea lui **I.**

Acestea sînt singurele instrucțiuni prin care programatorul poate comunica cu registrul **I.**

La inițializarea microprocesorului (aplicarea semnalului de comandă $\overline{\text{RESET}}$), în registrul **I** se înscrie valoarea **0.**

Continuînd explorarea fotografiei Im 0., să trecem la regiștri de uz general ai microprocesorului **Z80.**

1.1.2. Regiştri de uz general, accesibili utilizatorului

Regiştrii de uz general au menirea, aşa cum am văzut, să păstreze în imediata apropiere a „focarului" microprocesorului (circuit de comandă, ALU) date, care să poată fi accedate rapid, evitîndu-se pe cît se poate accesele la memorie, operaţii care sînt mult mai lente şi reduc astfel viteza de lucru a microprocesorului.

Într-o grupare semnificativă a figurii din Im. 0 distingem alături de regiştri deja prezentaţi (PC, SP, R, I) alte căsuţe simbolizate cu B, C, D, E, H, L, *regiştri de uz general primari*, B', C', D', E', H', L', *regiştri de uz general seundari*, precum şi IX respectiv IY care sînt *regiştri de index*.

Regiştrii B şi C : sînt *regiştri generali* de *8 bit*. Există o multitudine de instrucţiuni (de transfer, aritmetice, logice, de rotire, etc.) care tratează conţinutul acestor regiştri. Ei nu au fost figuraţi întîmplător pe aceeaşi linie : în anumite condiţii ei se pot ataşa, formînd un *registru pereche* BC, avînd lungimea de *16* bit. În această situaţie B este octetul *cel mai semnificativ*, iar C octetul *cel mai puţin semnificativ*. Există bineînţeles instrucţiuni care tratează perechea BC ca un registru de *16 bit* (instrucţiuni de transfer, instrucţiuni aritmetice).

Regiştrii D şi E : se caracterizează prin aceleaşi trăsături ca şi regiştri B şi C. Atunci cînd perechea DE formează un *registru dublu* de *16 bit*, D ocupă *octetul superior* iar E *octetul inferior*.

Regiştrii H şi L : *diferă* de cei anteriori B, C, D, E doar prin faptul că sînt implicaţi într-un număr mai mare de instrucţiuni, *avînd astfel un grad de funcţionalitate sporit faţă* de BC şi respectiv DE. Atunci cînd ei formează o *pereche* HL, registrul H ocupă *octetul superior*, iar L *cel inferior*. Pentru ca cititorul să poată sesiza de pe acuma plusul de funcţionalitate menţionat, prezentăm cîteva elemente legate de funcţionarea microprocesorului, în avans. Ştim că operaţiile aritmetice şi logice lucrează cu doi operanzi. Unul din aceştia va fi obligatoriu locat în registrul acumulator A. Celălalt se află de obicei într-unul din regiştri de uz general B, C, D, E, H, L. Există însă şi posibilitatea de a loca cel de al doilea operand în memorie, la o adresă oarecare. Pentru a efectua o operaţie aritmetică sau logică între registrul A şi conţinutul celulei de memorie respective, adresa celulei se va înscrie în registrul dublu HL. *Perechea* HL *devine astfel principalul instrument de adresare indirectă a unor operanzi locaţi în memorie la adrese cunoscute*. Nu există în schimb operaţii aritmetice/logice care să folosească pentru adresarea operandului locat în memorie conţinutul regiştrilor dubli BC şi DE. Se pot folosi în schimb regiştrii IX şi IY.

Regiştrii index IX şi IY : sînt prin definiţia dată de producători, regiştri de *16 bit*. Ei sînt *dedicaţi* eminamente *pentru stocarea unor adrese de memorie*. Faptul că există instrucţiuni care tratează aceşti regiştri ca şi regiştri de *8 bit*, este ascuns sub voalul tăcerii de către fabricanţi. Îi vom trata deci ca regiştri de *16 bit*. (Pentru informarea cititorului vom prezenta şi trăsăturile tainice ale acestor regiştri în cap. G (COMPLEMENTE). Regiştri IX şi IY au fost gîndiţi să fie utilizaţi atunci cînd trebuiesc efectuate *operaţii aritmetice/logice asupra unor cîmpuri de date*, date ce sînt aşezate la adrese succesive de memorie, formînd astfel un *tabel*. *Regiştrii index* nu vor conţine adresa celulei care se doreşte a fi tratată.

Ei *conţin adresa de început a tabelei de date. Data dorită se va localiza adăugînd la adresa de început a tabelei* **(adresa de bază),** *indicele* **(numărul de ordine)** *datei respective.* Indicele datei dorite se specifică explicit în enunţul instrucţiunii, valoarea lui fiind adunată în cursul execuţiei instrucţiunii la adresa de bază, conţinută în registrul index **IX** sau **IY**, formîndu-se astfel adresa efectivă a celulei de memorie dorite. Spre exemplificare să considerăm instrucţiunea :

ADD A,(IX + 5)

ea adună conţinutul registrului **A** cu cel al celui de-al cincilea element din tabelul de date, care începe la adresa conţinută în registrul **IX.** Conţinutul registrului **IX** nu se modifică pe parcursul execuţiei acestei instrucţiuni. Datele din tabelele adresate cu registrul index vor fi date cu lungimea de 1 octet. *Numărul maxim de elemente ce pot fi reperate cu o adresă de bază este de 256,* datorită faptului că indicele care apare în cîmpul instrucţiunii are lungimea de 1 octet. Programatorul dispune bineînţeles de instrucţiuni de iniţializare, salvare şi restaurare a regiştrilor index *IX* şi *IY.* Ba chiar mai mult, ei pot fi folosiţi şi pentru efectuarea unor operaţii aritmetice de 16 bit, înlesnindu-se astfel calculul unor adrese de bază, prin program. Această facilitate suplimentară îi îndeamnă pe mulţi programatori, să folosească regiştri dubli **IX** şi **IY** în alt scop decît cel pentru care ei au fost gîndiţi, ceea ce desigur nu este de condamnat. Astfel regiştri IX şi IY sînt des utilizaţi ca elemente de memorare a unor valori numerice de 16 bit. Amintim că *tehnica de adresare* a unor operanzi locaţi în memorie, folosind regiştri **IX** şi **IY** se numeşte *adresare indexată.* Vom detalia această tehnică în paragraful **„Tehnici de adresare".**

Regiştri secundari B', C', D', E', H', L'.

Ca şi în cazul regiştrilor speciali **A** şi **F,** unde am întîlnit deja regiştri secundari **A'** respectiv **F',** şi regiştri de uz general au cîte o dublură : regiştri secundari **B', C', D', E', H', L'.**

Conţinutul regiştrilor primari şi a celor secundari poate fi interschimbat concomitent, printr-o singură instrucţiune :

EXX

Toate instrucţiunile care implică regiştri de uz general **B, C, D, E, H, L** *acţionează asupra conţinutului* **regiştrilor primari.** Pentru a putea opera cu conţinutul regiştrilor secundari, el trebuie transferat în regiştri primari, folosind instrucţiunea **EXX.**

1.1.3. Regiştri de manevră, transparenţi utilizatorului

În interiorul oricărui microprocesor *există* şi anumiţi *regiştri care nu sînt folosiţi în mod explicit* de către instrucţiunile acelui microprocesor. În fotografia din Im. 0 am notat aceşti regiştri cu **T, T,W şi Z.** Ei sînt *regiştri de manevră,* transparenţi utilizatorului.

Regiştri W şi Z : sînt regiştri de cîte *8 bit* care pot fi trataţi şi ca *regiştri dubli de 16 bit.*

Exemplificăm necesitatea existenței acestor regiștri analizînd o instrucțiune simplă a microprocesorului Z80.

JP 1234H

Aceasta este o **instrucțiune de salt necondiționat (jump).** După efectuarea ei microprocesorul va executa secvența de cod (programul) care începe la adresa hexazecimală **1234.** Instrucțiunea se codifică pe 3 octeți :

C3H 34H 12H

Cei trei octeți ocupă în memorie locații succesive în ordinea enunțată. **C3H** reprezintă *codul instrucțiunii* de salt necondiționat (jump) ; **34H** reprezintă *octetul inferior* (mai puțin semnificativ) al adresei de salt ; **12H** reprezintă *octetul superior* (cel mai semnificativ) al adresei de salt. Pentru a fi mai expliciți să specificăm o adresă oarecare la care este locată instrucțiunea : fie această adresă **100H :**

0100H — C3H

0101H — 34H

0102H — 12H

În momentul în care începe execuția instrucțiunii, contorul program **PC** conține valoarea **0100H.** După citirea codului instrucțiunii, **C3H, PC** este incrementat, conținutul lui devenind **0101H.** De la această adresă se citește octetul inferior **(34H)** al adresei de salt. **PC** se incrementează din nou, indicînd **0102H.**

Ei, acu-i acu ! Unde se depune acest octet ? Dacă el s-ar depune în octetul inferior al contorului program **PC,** atunci conținutul acestuia ar deveni : **0134H,** și *ar fi* deci *imposibilă citirea octetului superior* al adresei de salt, octet care se află în memorie la adresa **0102H.** De aceea *este necesar* ca în interiorul microprocesorului să existe *un registru tampon.* Fie acest registru registrul **Z.** Aici se depune octetul inferior al adresei de salt. În următorul ciclu de funcționare microprocesorul citește de la adresa **0102H** octetul superior al adresei de salt **12H,** și îl depune în registrul tampon **W.** Odată efectuate aceste manevre conținutul registrului **WZ** se transferă în **PC,** efecutîndu-se saltul dorit, la adresa **1234H.**

Regiștri temporari T : sînt *regiștri de intrare* în unitatea aritmetică/logică **(ALU).** În Im 0. acești regiștri au același nume **T,** datorită faptului că n-am dorit să încărcăm utilizatorul programului **VISIBLE—Z80** cu elemente care sînt totuși secundare ca importanță. Dorind să sugerăm cititorului o atitudine critică, care să-l determine a căuta cauzele oricărui fenomen, vom argumenta și pentru necesitatea existenței a doi regiștri tampon la intrarea în **ALU.** În *literatura de firmă,* schema bloc a microprocesorului apare cu *un singur registru tampon la intrarea* **ALU,** cealaltă intrare fiind legată la acumulatorul **A.** La prima vedere această afirmație pare a fi corectă, noi știind deja că toate operațiile **ALU** folosesc unul din operanzi provenind din **A.** Intercalarea unui registru tampon între **A** și **ALU** pare a fi inutilă. Dar *analiza detailată* a instrucțiunilor microprocesorului, *demonstrează contrariul.*

Să considerăm exemplul deja prezentat :

ADD A,(IX+5)

Fie conținutul registrului **IX** o valoare oarecare (număr hexazecimal cu patru cifre semnificative = 16 bit), de exemplu **44FEH.** Înainte de a se efectua adunarea stipulată în instrucțiune, trebuie citit **operandul 2** din memorie, de la adresa **IX+5 = 44FEH+5.** Suma dintre adresa de bază **44FEH** și indicele cerut **5** va fi efectuată de către microprocesor tot prin **ALU.** Această etapă nu are însă voie să afecteze conținutul registrului **A,** fiindcă astfel am compromite din start obiectivul final al instrucțiunii : anume, să calculeze suma dintre acumulator și celula de memorie avînd adresa **44FEH + 5 = 4502H.**

Iată deci o operație aritmetică intermediară, cea de calcul a adresei elementului indexat, care se face, fără participarea acumulatorului **A.** *Registrul tampon din partea stîngă* va conține de exemplu indicele **5** și cel din *dreapta* octetul inferior al adresei de bază : **FEH.**

Astfel s-a demonstrat necesitatea prezenței unui registru tampon între **A** și **ALU.**

De ce nu se poate efectua adunarea a doi regiștri

ADD A,B

fără existența registrului tampon **T** din dreapta, este o întrebare la care răspunsul ar trebui să intre în detalii hardware care depășesc obiectivul acestei cărți.

Registrul tampon de date DB (Data Buffer) este un *registru bidirecțional* de *8 bit,* care delimitează interiorul microprocesorului, de lumea externă. El este necesar în primul rînd pentru a menține datele ce sînt trimise din celulele registrilor interni spre memorie, sau spre periferice.

Registrul tampon de adrese AB (Address Buffer) *izolează* pe de o parte *magistrala externă* de adrese *de cea internă.* În afară de aceasta el are un rol de *memorare temporară* a unor adrese, rol pe care l-am explicitat la relevarea modelului funcțional al unui microprocesor ipotetic.

Sperînd că nu v-am epuizat total cu această prezentare, închidem aici lista regiștrilor interni ai microprocesorului **Z80.**

1.2. Circuitul de comandă

Acest bloc funcțional, notat în Im. 0 cu **Command Circuit,** este *inima* și *creierul oricărei unități centrale* de calculator și implicit al microprocesoarelor. Sarcina lui este *să guverneze toate activitățile* microprocesorului, începînd din momentul în care acestuia i se aplică tensiune și semnal de tact.

Circuitul de comandă generează toate semnalele de comandă pentru activarea secvențială a tuturor elementelor interne și a „lumii" externe, din microcalculator. În linii mari, în activitatea lui se disting următoarele faze majore : *citește* din memorie codul instrucțiunii de executat, îl *decodifică. Înțelegînd* semnificația codului citit programează toate activitățile interne și externe pentru a *duce la bun sfîrșit comanda primită.* Dacă aceste activități sînt multe sau puține, o lăsăm pe seama cititorului. Amintim doar că *cea mai scurtă* instrucțiune se execută în **4** tacți procesor (ex. : transferul între regiștri : **LD A, B, LD D, E,** etc.), pe cînd

cea mai lungă necesită **23** de stări de tact distincte, fiecare cu semnificația ei (ex. : ștergerea unui bit dintr-o locație de memorie adresată printr-un registru index : **RES 3, (IX + IND)).**

În blocul funcțional al circuitului de comandă am inclus și *bistabilii de validare/inhibare a sistemului de întreruperi* **IFF1** și **IFF2,** precum și *registrul de 2 bit* al **modulului de întreruperi** folosit. *Dacă* **IFF1** *și* **IFF2** *au valoarea* **1,** *atunci sistemul de întreruperi este validat, microprocesorul acceptînd cererile de întrerupere sosite din lumea externă.* Dacă ele au valoarea zero, sistemul de întreruperi este inhibat, cererile de întrerupere fiind neglijate. În *registrul* **MOD** se stochează modul de întrerupere selectat prin una din instrucțiunile de comandă **IM 0, IM 1, IM 2.** Asupra rolului acestor elemente funcționale și a funcționării lor vom reveni în aventurile pe care le vom întreprinde cu **VISIBLE—Z80.**

Proiectarea și analizarea unui astfel de circuit electronic este de neconceput fără folosirea intensivă a unor tehnici de proiectare asistată de calculator și fără tehnologie de vîrf. Nu vom insista asupra detaliilor constructive și de funcționare a circuitului de comandă pentru a nu devia de la obiectul cărții. Amintim doar că orice relatare pe care am face-o referitor la acest aspect, ar fi pur teoretică, firmele producătoare nepublicînd mai nimic despre această structură funcțională a microprocesorului.

1.3. Unitatea aritmetică/logică

Așa cum rezultă din relatările anterioare, **ALU** este un *bloc funcțional al microprocesoarelor,* destinat să efectueze *operațiile aritmetice și logice.* Unitatea aritmetică/logică a microprocesorului **Z80** este de tip paralel, avînd *lățimea de 8 bit. Această unitate centrală poate executa două operații aritmetice : adunarea și scăderea a două numere binare de cîte 8 bit.* Operațiile logice acționează de asemenea pe cuvinte de cîte 8 bit. Ele sînt **ȘI, SAU, SAU EXCLUSIV, COMPARAȚIE** și **COMPLEMENTARE** (față de 1 și 2).

Unul din operanzii implicați în operațiile **ALU,** va proveni totdeauna din registrul acumulator **A.** Odată cu generarea rezultatului operațiilor aritmetice/logice, se poziționează și indicatorii de condiție ai registrului **F.**

Setul de instrucțiuni aritmetice și logice pare a fi *sărac pentru* cel care dorește să execute *calcule științifice* folosind instrucțiunile enunțate. Pentru începători pare puțin probabil să calculezi logaritmul unui număr real cu o precizie de 16 cifre semnificative, atunci cînd dispui de o unitate aritmetică care știe doar să adune și să scadă două numere întregi avînd valori cuprinse în limitele [0,255]. Ținem să-i liniștim afirmînd că este posibil, trebuie să cunoști însă bazele analizei numerice pentru a reduce calculul unui logaritm la adunări și scăderi, și să ai iscusința de a transpune acești algoritmi în programe scrise cu ajutorul instrucțiunilor limitate ale microprocesoarelor.

ALU *efectuează scăderile transformînd scăderile în adunări.* Numerele de scăzut (negative) sînt transformate în *complementul lor față de 2.*

Circuitele electronice care realizează funcțiile **ALU** sînt de tip combinațional, fiind mult mai simple decît cele din circuitul de comandă a microprocesorului.

1.4. Magistralele interne

Magistrala de 8 bit.

Știind că lățimea majorității regiștrilor din interiorul microprocesorului **Z80** este de un octet, deducem că în interiorul lui trebuie să existe o magistrală avînd lățimea de *8 bit* pentru a se putea efectua transferurile interne. La această magistrală de date care începe la bufferul de date **DB** se leagă registrul instrucțiune **IR**, acumulatorul **A**, regiștri tampon **T** de la intrarea **ALU**, ieșirea **ALU**, precum și toți ceilalți regiștri interni de 8 bit.

Magistrala de 16 bit.

Adresele de memorie pe care microprocesorul le generează au lățimea de *16 bit*. În interiorul microprocesorului ele se vehiculează pe o *magistrală locală* care se termină la registrul tampon pentru adrese **AB (Address Buffer)**.

Din studiul detaliat al funcționării microprocesorului rezultă că toți regiștri **B, C, D, E, H, L, IX, IY, SP, PC** și **WZ** au cîte **2 ieșiri** : *una* din ieșiri este racordată *la* magistrala de *8 bit*, folosindu-se pentru transferul de date ; *cealaltă* ieșire a lor este legată la octetul superior sau inferior al *magistralei de 16 bit*, permițînd astfel generarea unor adrese de 16 bit pentru adresarea memoriei externe.

Semnalele de comandă interne

Pentru a putea activa în ordinea dorită, fiecare element funcțional intern, circuitul de comandă al microprocesorului dispune de o multitudine de semnale de comandă, semnale care ajung la toate părțile lui componente. Ele nu au fost reprezentate în modelul din Im 0, numărul lor mare nepermițînd acest lucru. Oricum este suficient să știm de existența lor. Nefiind nominalizate, aceste semnale vor fi grupate sub numele generic de semnale de comandă interne.

2

SEMNALELE EXTERNE

După ce am cunoscut structura internă a microprocesorului, este momentul să prezentăm semnalele prin care el comunică cu lumea externă. În fig. 2.1. redăm semnalele externe ale microprocesorului **Z80**, grupate funcțional.

Distingem 3 clase mari de semnale : **magistrala de date, magistrala de adrese** și **magistrala de comenzi.**

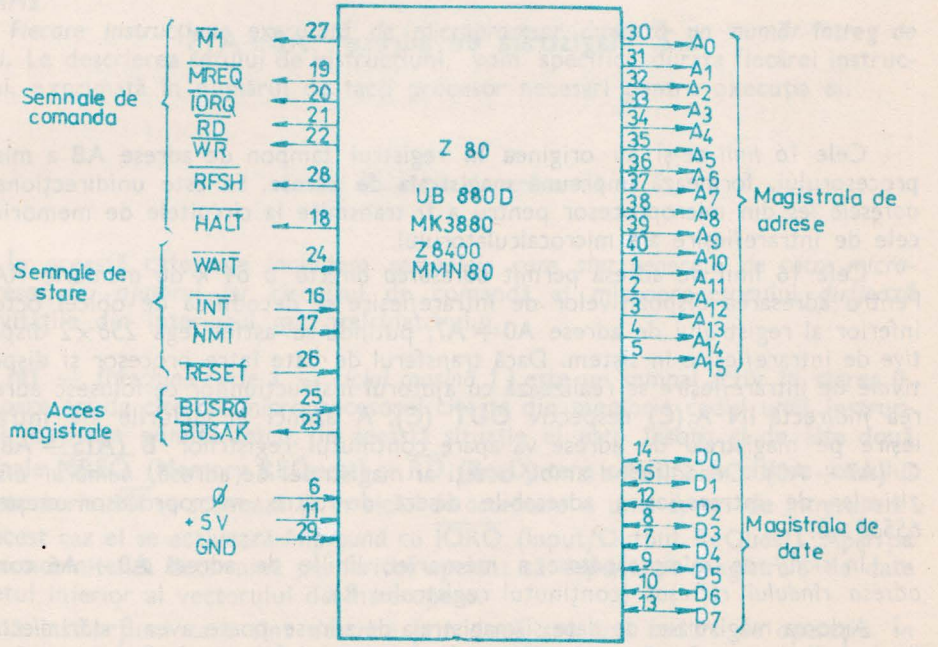


Fig. 2.1. Semnalele externe ale microprocesorului Z80

2.1. Magistrala de date : D0 ÷ D7.

Cele 8 linii care pornesc din registrul tampon de date **DB** al microprocesorului sînt folosite pentru a asigura transferul de date între microprocesor, memorie și dispozitivele de intrare/ieșire. Împreună, ele formează **magistrala de date** a microprocesorului, notată cu **D0 ÷ D7**. Magistrala de date a microprocesorului este *bidirecțională*. Pe ea *intră date* atunci cînd se efectuează o citire din memorie sau dispozitivele de intrare/ieșire, și *ies date la efectuarea* unei scrieri.

Magistrala este de tip *tri-state* : pe lîngă cele *două nivele logice active* (0 și 1), fiecare linie de date poate avea *o stare* inhibată, de *înaltă impedanță*. Această stare este utilă dacă în anumite momente de funcționare, magistrala de date trebuie cedată de către microprocesor unei alte componente din microcalculator. Un exemplu în acest sens îl poate constitui efectuarea unui transfer de date, direct de la un dispozitiv de intrare/ieșire în memorie (sau invers) procedeu numit **DMA** (Direct Memory Acces — acces direct la memorie).

Magistrala de date este *neinversată* : biții de valoare 1 vor apare cu nivele de tensiune ridicată (3 ÷ 4 V), pe cînd cei de valoare 0 vor fi materializați prin nivele de tensiune scăzute (0,1 ÷ 0,8 V). **D0** este *bitul cel mai puțin semnificativ* al octeților ce circulă *pe magistrala de date*.

2.2. Magistrala de adrese : A0 ÷ A15

Cele *16 linii* ce-și au originea în registrul tampon de adrese **AB** a microprocesorului, formează împreună **magistrala de adrese**. Ea este unidirecțională : *adresele ies* din microprocesor pentru a fi transmise la circuitele de memorie și cele de intrare/ieșire ale microcalculatorului.

Cele 16 linii de adresă permit *adresarea directă a 64 k* de memorie RAM. Pentru adresarea dispozitivelor de intrare/ieșire se decodifică de obicei octetul inferior al registrului de adrese **A0 ÷ A7**, putîndu-se astfel lega 256×2 *dispozitive* de intrare/ieșire în sistem. Dacă transferul de date între procesor și dispozitivele de intrare/ieșire se realizează cu ajutorul instrucțiunilor ce folosesc adresarea indirectă **IN A,(C)** respectiv **OUT (C), A** atunci în ciclurile de intrare/ieșire pe magistrala de adrese va apare conținutul regiștrilor **B (A15 ÷ A8)** și **C (A7 ÷ A0)**. Decodificînd ambii octeți ai magistralei de adrese, *numărul dispozitivelor* de intrare/ieșire adresabile direct de către microprocesor *crește la* 65536×2.

În *ciclul de reîmprospătare* a memoriei, liniile de adresă **A0 ÷ A6** conțin *adresa rîndului refresat* (conținutul registrului **R**).

Aidoma magistralei de date și magistrala de adrese poate avea *3 stări electrice* (logice) distincte : *2 stări active* (1 sau 0) și *o stare* inhibată, *de înaltă impedanță*. Menirea celei de a treia stări este identică cu cea enunțată la magistrala de date.

Linia **A0** conține bitul *cel mai puțin semnificativ al adreselor*.

2.3. Magistrala de comenzi

Acest nume este un *nume generic*, căci spre deosebire de celelalte două magistrale, în care liniile fiecăreia aveau semnificații similare, magistrala de comenzi este *reuniunea unor semnale individuale* de intrare sau de ieșire din microprocesor, *avînd fiecare un rol aparte.*

Le clasificăm totuși în trei grupe :

— **semnalul de tact**
— **semnale de comandă**
— **semnale de stare**

2.3.1. Semnalul de tact (Ø)

Este un *semnal* **TTL**, constituind *elementul motor* al microprocesorului. Toate evenimentele interne ale microprocesorului apar sincronizate cu acest semnal. *Frecvența* semnalului *poate varia* în limite destul de largi, interesul nostru fiind însă acela de a lucra cu frecvențe cît mai mari ale semnalului de ceas, datorită faptului că *viteza de execuție a instrucțiunilor este direct proporțională cu frecvența tactului Ø.*

Primele microprocesoare **Z80** au avut ca *limită superioară* a frecvenței de tact 2,5 *MHz*. Ulterior au apărut *variante mai rapide* : **Z80A** *cu 4 MHz și* **Z80B** *cu 6 MHz.*

Fiecare instrucțiune executată de microprocesor durează un număr întreg de tacți. La descrierea setului de instrucțiuni, vom specifica durata fiecărei instrucțiuni, exprimată în numărul de tacți procesor necesari pentru execuția ei.

2.3.2. Semnale de comandă

În această categorie includem *semnalele care sînt generate de către microprocesor. Cu ajutorul lor*, circuitul de comandă al microprocesorului *dirijează* activitățile din interiorul microcalculatorului.

$\overline{M1}$ — (*Machine cycle* **1** — *ciclul mașină 1*) este un semnal activ în starea **0**. El apare ori de cîte ori microprocesorul citește din memorie codul unei instrucțiuni, operație numită fetch. În această situație el este însoțit și de alte două semnale $\overline{MREQ}$ (Memory REQuest) și $\overline{RD}$ (ReaD) care concură la citirea codului din memorie. $\overline{M1}$ se activează și în ciclul de *acceptare* a unei cereri de *întrerupere*. În acest caz el se activează *împreună* cu $\overline{IORQ}$ (Input/Output ReQuest). Apariția lor concomitentă determină perifericul apelant să depună pe magistrala de date octetul inferior al vectorului de întrerupere.

Există instrucțiuni care sînt descrise prin două sau trei coduri de operație. În cursul execuției acestora $\overline{M1}$ *se va activa la citirea fiecărui* octet cod de operație. $\overline{M1}$ poate trece în starea de înaltă impedanță *(tri-state)*.

MREQ — (*Memory* **REQ***uest — cerere de acces la memorie*), este un semnal activ în starea **0**. El se activează ori de cîte ori microprocesorul dorește să efectueze un acces la memorie, fie pentru citirea, fie pentru înscrierea ei.
MREQ se activează și pe parcursul ciclului de *reîmprospătare* a memoriilor dinamice, *acompaniat* de semnalul **RFSH**.
În momentul apariției acestui semnal magistrala de adrese conține pe liniile **A0 ÷ A15** adresa celulei de memorie la care se va efectua accesul.
MREQ poate trece în starea de înaltă impedanță *(tri-state)*.

RD — (*Rea***D** — *citește*) este un semnal activ în starea **0**. El este activat atunci cînd microprocesorul execută citirea unui octet, din memorie sau de la un dispozitiv de intrare. Elementul apelat (celula de memorie sau dispozitivul de intrare) va transfera la apariția acestui semnal conținutul lui pe magistrala de date. Sursa datei de citit se stabilește din starea semnalelor **MREQ** și **IORQ**.
RD poate trece în starea de înaltă impedanță *(tri-state)*.

WR — (**WR***ite — scrie*) este un semnal activ în starea **0**. El se activează atunci cînd microprocesorul urmează să scrie un octet în memorie sau într-un dispozitiv de ieșire. Destinația octetului transferat se stabilește din starea semnalelor **MREQ** și **IORQ**. Înainte de activarea semnalului **WR** microprocesorul va depune pe magistrala de date octetul de transferat. Datele fiind stabile în momentul apariției semnalului de scriere **WR**, el poate fi folosit (frontul său căzător) pentru înscrierea lor în celule de memorie sau dispozitivul de ieșire dorit.
WR poate trece în starea de înaltă impedanță *(tri-state)*.

IORQ — (*Input/Output* **ReQ***uest — cerere de acces la dispozitivul de intrare/ieșire*) este un semnal activ în starea **0**. El este *acompaniat* de unul din semnalele **RD** sau **WR**, împreună cu care determină tipul de acces efectuat la dispozitivele de intrare/ieșire. În ciclul de *acceptare* a cererii de *întrerupere*, **IORQ** se activează *împreună* cu **M1**. Prezența celor două semnale semnifică acceptarea unei cereri de întrerupere. **IORQ** poate trece în starea de înaltă impedanță *(tri-state)*.

RFSH — (**ReFreSH** — *reîmprospătare*) este un semnal activ în starea **0**. El se generează în *ciclul de reîmprospătare* a memoriilor dinamice. Prezența acestui semnal, activat, înseamnă că pe liniile **A0 ÷ A6** s-a depus conținutul registrului **R**. La apariția semnalului **RFSH** aceste linii nefiind încă stabilizate, se recomandă să se folosească semnalul **MREQ**, care apare în acest ciclu puțin mai tîrziu, pentru declanșarea accesului de reîmprospătare a memoriei.

HALT — (*Halt — oprește*) este un semnal activ în starea **0**, prin care microprocesorul aduce la cunoștința sistemului, faptul că a executat o instrucțiune **HALT** și este oprit. Din această stare, microprocesorul nu poate fi scos decît printr-o întrerupere **INT** (dacă **IFF1** = 1), o întrerupere nemascabilă **NMI**, sau o reinițializare **RESET**. În realitate microprocesorul nu stă pe loc, ci execută în continuare instrucțiuni **NOP** (**No OP**eration) pentru a asigura *reîmprospătarea* unei eventuale memorii dinamice din sistem.

Dacă ieşirea din starea **HALT** s-a făcut printr-o întrerupere, (mascabilă sau nemascabilă), după tratarea ei, execuţia programului va continua de la instrucţiunea locată la adresa următoare instrucţiunii **HALT**.

BUSAK (BUS AcK*nowledge — acceptarea cererii de magistrală*) este un semnal activ în starea **0**, prin care microprocesorul confirmă acceptarea unei cereri **BUSRQ**, de cedare a magistralelor. Semnalul **BUSAK** apare totdeauna după terminarea unui ciclu maşină, după ce microprocesorul şi-a pus în stare de înaltă impedanţă magistralele (date, adrese, comenzi).
BUSAK *nu trece niciodată în starea de înaltă impedanţă.*

2.3.3. Semnale de stare

O fi microprocesorul „creierul şi conducătorul" vieţii interne a unui microcalculator, dar oricît de atotputernic ar fi, nu-şi poate permite să dea comenzi orbeşte, fără să-i pese de răspunsul celorlalte componente din sistem. Uneori va trebui să-şi încetinească ritmul, alteori va fi întrerupt în marşul său, părăsind temporar drumul pe care se afla, pentru a se putea îngriji la cerere de subalternii săi, ba chiar mai mult, în anumite circumstanţe va trebui să stea deoparte, lăsîndu-i şi pe alţii să vorbească — să preia comanda — dacă există activităţi pe care ei le pot efectua mai rapid şi mai eficient.

În ultimă instanţă, dacă în munca sa microprocesorul s-a pierdut pe drum, pe cărările accidentate ale unui program insuficient de bine pus la punct, trebuie să existe posibilitatea de a-l readuce cu „picioarele pe pămînt", forţîndu-l să ia totul de la început.

Microprocesorul culege informaţiile necesare pentru luarea unor astfel de decizii, prin intermediul unor semnale de intrare numite de către noi, semnale de stare.

Să le cunoaştem :

WAIT — *(Aşteaptă)* este un *„strigăt"* emis de elementul apelat (celulă de memorie sau dispozitiv de intrare/ieşire) prin care acesta semnalizează microprocesorului faptul că merge prea repede, şi el, colaboratorul, nu poate ţine pasul. În fiecare ciclu de memorie sau de intrare/ieşire, după lansarea comenzii (activarea semnalelor **MREQ** sau **IORQ** şi **RD** sau **WR**), circuitul de comandă al microprocesorului analizează starea liniei **WAIT**. Dacă ea este **activă** (valoare **0**) atunci se inserează cicluri de tact suplimentare de aşteptare, pînă cînd elementul apelat îşi termină treaba. La dispariţia semnalului **WAIT**, microprocesorul continuă execuţia instrucţiunii ca şi cum nu s-ar fi întîmplat nimic. Acest semnal permite ca şi un microprocesor ori cît de rapid să poată coexista şi colabora cu circuite de memorie sau de intrare/ieşire lente. Dacă în sistem coexistă elemente rapide (ex. : circuite RAM cu timp de acces de 100 ÷ 200 ns.) şi lente (ex. : memorie de ferită cu timp de acces de ordinul microsecundelor), *semnalul* **WAIT** *adaptează viteza de rulare a microprocesorului la posibilităţile concrete ale fiecăruia.*

Specificăm totuşi, că dacă în sistem există memorii RAM dinamice care aşteaptă să fie reîmprospătate de către microprocesor, atunci aşteptarea cauzată de semnalul $\overline{\text{WAIT}}$ *nu poate fi oricît de lungă, fiindcă altfel s-ar risca pierderea informaţiei stocate în DRAM.*

$\overline{\text{INT}}$ (INT*errupt — întrerupere*) este *„petiţia"* depusă de un element din sistem (de obicei un dispozitiv de intrare/ieşire), prin care se solicită participarea microprocesorului la rezolvarea unei situaţii noi, create de apariţia unui eveniment extern. Dacă sistemul de întreruperi al microprocesorului este validat, atunci **activarea** semnalului $\overline{\text{INT}}$ (valoare **0**), va declanşa abandonarea programului în curs de execuţie şi microprocesorul efectuează un salt la o rutină în care tratează cererea de întrerupere. După terminarea rutinei de întrerupere microprocesorul se întoarce în programul abandonat, reluîndu-i execuţia din punctul în care a fost abandonat. Cererea de întrerupere este acceptată numai după terminarea ultimului ciclu maşină a instrucţiunii în curs de execuţie. *Cererea de întrerupere este ignorată* dacă microprocesorul este în *stare de aşteptare* ($\overline{\text{WAIT}}$), *dacă este suspendat* ($\overline{\text{BUSAK}}$), sau dacă *sistemul de întreruperi este inhibat* (bistabilul **IFF1=0**). Dacă cererea de întrerupere se acceptă (apar concomitent semnalele $\overline{\text{M1}}$ şi $\overline{\text{IORQ}}$), saltul la începutul rutinei de tratare a întreruperii se va efectua în diverse moduri în funcţie de regimul programat **IM 0, IM 1** sau **IM 2** (Interrupt Mode).

$\overline{\text{NMI}}$ — (*Non Mascable Interrupt — întrerupere nemascabilă*) este *„pumnul în masă"* bătut de un element din sistem la apariţia unui fenomen care trebuie deservit imediat, indiferent de importanţa programului în curs de execuţie. Pe cînd cererile de întrerupere sosite pe linia $\overline{\text{INT}}$ puteau fi desconsiderate — prin inhibarea pe cale software a sistemului de întreruperi, *cererile* sosite pe linia $\overline{\text{NMI}}$ *vor fi obligatoriu deservite*, în cel mai scurt timp posibil (la terminarea instrucţiunii în curs de execuţie).
Există şi două cazuri în care cererea $\overline{\text{NMI}}$ rămîne fără răspuns : în situaţiile în care microprocesorul este imobilizat (stare **WAIT** continuă, sau $\overline{\text{BUSAK}}$).

$\overline{\text{NMI}}$ va întrerupe chiar şi un program de tratare a unei întreruperi sosite pe linia $\overline{\text{INT}}$.

La acceptarea cererii de întrerupere nemascabilă, microprocesorul salvează conţinutul contorului program **PC** pe stivă şi execută un salt necondiţionat la adresa 0066H, unde trebuie să înceapă rutina de tratare a întreruperilor nemascabile.

După terminarea acestui program se poate restaura vechea valoare a **PC** de pe stivă, pentru a relua programul abandonat. $\overline{\text{NMI}}$ este singurul semnal al microprocesorului **Z80** care datorită importanţei sale (trebuie evitată posibilitatea trecerii neobservate a unei cereri $\overline{\text{NMI}}$) nu se activează pe nivel, ci pe front.

Cererea $\overline{\text{NMI}}$ se înregistrează la apariţia unui front scăzător pe acest semnal, cerere care va persista pînă la acceptare (sfîrşitul instrucţiunii în curs), chiar dacă între timp cererea în sine a dispărut.

Evenimente importante care merită să fie semnalate prin $\overline{\text{NMI}}$ sînt : avertisment la căderea tensiunii sau intervenţie operator.

$\overline{\text{BUSRQ}}$ (BUS — ReQuest — *cerere de magistrală*) este *„piciorul pus în prag"* de către un element din sistem, manevră prin care se suspendă microprocesorul. Acesta din urmă va termina ciclul mașină în curs de execuție, după care emite semnalul de acceptare a cererii : $\overline{\text{BUSAK}}$, și va „sta deoparte" cu toate magistralele puse în starea de înaltă impedanță (cedate). În această situație dispozitivul care a suspendat microprocesorul va prelua comanda magistralelor din microcalculator. Această facilitate poate fi folosită pentru a realiza un sistem cu două procesoare diferite, din care doar unul să fie activ, sau pentru a realiza un dispozitiv cu acces direct la memorie. Un dispozitiv DMA poate efectua un transfer mai rapid între un echipament periferic și memoria microcalculatorului, decît ar putea-o face microprocesorul prin program. $\overline{\text{BUSRQ}}$ este activ în starea 0.

$\overline{\text{RESET}}$ (*Reinițializare*) este *„mîna de ajutor"* acordată de către operator microprocesorului (sau de către proiectanții microprocesorului — operatorului ? !). Acest semnal permite reinițializarea microprocesorului. La *activarea* lui (valoarea **0**) se întîmplă următoarele evenimente în interiorul microprocesorului :

— se șterge contorul program	PC=0000H
— se inhibă sistemul de întreruperi	IFF1=IFF2=0
— se stabilește regimul de întreruperi	IM 0
— se inițializează regiștrii I și R	I=R=00H

Semnalul $\overline{\text{RESET}}$ se va aplica microprocesorului la „trezire" (după pornirea de la butonul rețea al unui dispozitiv cu microprocesor) și oricînd execuția unui program „s-a rătăcit" sau a intrat într-o buclă infinită.

Pentru a fi luat în considerare, semnalul $\overline{\text{RESET}}$ trebuie să fie activ minimum 3 tacți procesor. Pe durata aplicării semnalului $\overline{\text{RESET}}$=0 magistralele de date și adrese ale microprocesorului Z80 trec în stare de înaltă impedanță (tri-state) iar semnalele de comandă în stare inactivă (1 logic). În acest răstimp „nimeni" nu lucrează, deci nu vor exista ciclurile de reîmprospătare a memoriei ($\overline{\text{RFSH}}$).

Orice încercare de a realiza un sistem în care aceste legături inverse, feed—back, lipsesc ar fi sortită eșecului. Dacă aceste semnale nu există, sau dacă ele nu sînt luate în considerare nu poate rezulta nimic altceva decît un sistem rigid ce nu se adaptează la condițiile schimbătoare ale mediului său înconjurător, un sistem în care ideea de siguranță primează asupra criteriilor de performanță.

3

SETUL DE INSTRUCȚIUNI

Totalitatea instrucțiunilor pe care un microprocesor le poate recunoaște și executa, formează setul de instrucțiuni al microprocesorului respectiv.

Dacă am fi puși în situația — și în realitate ni se întîmplă destul de des — să comparăm două microprocesoare pentru a-l alege pe cel mai performant, sau pe cel care se pretează mai bine pentru rezolvarea problemei noastre, nu vom recurge la o analiză comparativă a seturilor lor de instrucțiuni. Pe lîngă spațiul de adresare, lățimea magistralei de adrese, numărul regiștrilor interni și frecvența maximă de tact, calitatea setului de instrucțiuni este aceea care determină în ultimă instanță performanța unei unități centrale de calculator și implicit și pe cea a unui microprocesor.

Cu cît setul de instrucțiuni este mai mare, cu cît el este mai variat — distingîndu-se cît mai multe clase de instrucțiuni —, cu cît tehnicile de adresare folosite sînt mai bogate, cu atît mai multe tipuri de probleme se vor putea rezolva în mod flexibil și eficient, cu atît mai performant va fi microprocesorul în cauză.

Pentru a da un exemplu sugestiv în acest sens — rămînînd în lumea microprocesoarelor de 8 bit — vom compara în linii mari cei doi corifei, microprocesorul **I8080** al firmei Intel și **Z80** al firmei Zilog, analizînd seturile lor de instrucțiuni.

În primul rînd totalul instrucțiunilor : **I8080** recunoaște și execută **244** de instrucțiuni, pe cînd **Z80—696** declarate de către fabricanți și încă cîteva sute nedeclarate, a căror existență a fost descoperită întîmplător de unii utilizatori, sau dedusă logic din analiza codurilor, de către alții.

Dacă amintim că **Z80** este compatibil de sus în jos cu **I8080,** adică execută toate instrucțiunile acestuia în mod identic, fără nici un fel de modificări, atunci și această primă comparație ar fi suficientă pentru a-l alege ca cel mai bun.

Continuăm totuși comparativ, nu atît pentru a savura victoria repurtată de către alesul nostru, ci mai mult pentru a introduce cîteva noțiuni. noi.

● Pe cînd **I8080,** — exceptînd două instrucțiuni — operează numai pe octeți, **Z80** dispune de 240 instrucțiuni care tratează selectiv biții dintr-un octet și de 16 instrucțiuni care vehiculează blocuri de date, a căror lungime poate atinge valoarea de 65.536 octeți.

● Pe cînd **I8080** poate efectua doar patru tipuri de rotiri sau deplasări asupra unui octet — și el locat obligatoriu în registrul acumulator **A** — , **Z80** recunoaște 9+1 tipuri de astfel de instrucțiuni, și ele pot opera asupra conținutului

oricărui registru de uz general, ba chiar mai mult folosind tehnici de adresare speciale, chiar și asupra conținutului oricărei locații de memorie.

● Pe cînd **I8080** poate executa un singur tip de operație aritmetică (adunare) asupra unor regiștri dubli (cuvinte de 16 biți), **Z80** recunoaște trei tipuri de instrucțiuni de același fel, și efectuează pe lîngă adunare și scăderea.

● Pe cînd **I8080** poate calcula direct doar complementul față de 1 al unui octet, **Z80** ne oferă prin instrucțiunea sa **NEG** și complementul față de 2 al unui byte.

● În fine, pe cînd **I8080** execută toate transferurile la dispozitive de intrare/ieșire prin registrul acumulator **A**, **Z80** le poate realiza „via" oricare din registrele sale de uz general, ba chiar mai mult direct între memorie și dispozitivul de intrare/ieșire.

Trecînd la compararea tipurilor de adresare folosite, constatăm că microprocesorul **I8080** cunoaște 4 tipuri : implicit, imediat, indirect („via" registru) și direct absolut. **Z80** aduce în plus 3 noi tipuri de adresare : indexat, indirect („via" memorie) și direct relativ.

Tot la compararea tehnicilor de adresare amintim faptul că dacă **I8080**, prin „propria-i forță" fără alte resurse hardware constituite de utilizator, poate adresa 256 de porturi de ieșire și tot atîtea de intrare, **Z80** „vorbește" nemijlocit cu 2×65536 de asemenea dispozitive. Un număr imens, de altfel. Nu cunoaștem să se fi realizat undeva în lume un microsistem bazat pe un microprocesor de 8 biți (Z80), care să fi avut atîtea dispozitive de intrare/ieșire.

Conchidem prin a constata un lucru evident : misiunea noastră, de a-l alege pe cel mai performant dintre cele două microprocesoare, a fost în acest caz deosebit de ușoară. Probabil că dacă vom dori pe viitor să comparăm alte microprocesoare, vom avea mai multă bătaie de cap. De aceea, vom explicita cîteva din noțiunile deja folosite.

3.1. Formatul instrucțiunilor

Așa cum am arătat în capitolul introductiv, codurile instrucțiunilor direct executabile de către microprocesor nu sînt altceva decît numere binare. Fiecare cod declanșează, după decodificarea sa în circuitul de comandă al microprocesorului, o serie de activități ce converg spre efectuarea „poruncii" transmise de către acel cod. Se pune de asemenea întrebarea : ce lungime (cîți biți sau octeți) trebuie să aibă numărul binar (cuvîntul de cod) pentru a defini univoc o instrucțiune ? În cazul microprocesorului **I8080**, a cărui set de instrucțiuni numără 244 de comenzi posibile, 8 bit se dovedesc a fi suficienți căci cu ajutorul lor se pot scrie $2^8 = 256$ de numere binare distincte. Cele 696 de instrucțiuni declarate ale microprocesorului **Z80** necesită în schimb un cuvînt binar mai lung. 10 bit vor ajunge desigur : $2^{10} = 1024$ combinații posibile. Datorită faptului că structura microprocesorului este orientată pe byte (8 bit), și memoria care i se atașează, va conține cuvinte avînd lungimea de 1 octet. Ar fi inconvenabil, ba chiar foarte greu de realizat, o structură care să trateze coduri de 10 bit și date de 8 bit, cu atît mai mult cu cît ambele circulă pe aceeași cale : magistrala de date a microcalculatorului. De aceea pentru a codifica un număr mai mare de instrucțiuni

decît 256 s-a recurs la o tehnică specială : unele instrucțiuni vor fi reprezentate pe doi octeți.

În setul primar de coduri (numere cuprinse între 0—256) se nominalizează cîteva, ele urmînd să fie coduri rezervate. La citirea unui cod rezervat, microprocesorul va sesiza că pentru a înțelege porunca codificată el trebuie să mai citească un octet de cod din memorie. Fiecare cod rezervat mărește astfel totalul instrucțiunilor codificate cu 256. Dorind să fim riguroși, vom nota cu n numărul codurilor rezervate adoptate. Atunci totalul instrucțiunilor codificabile prin această tehnică devine :

$$N = 256 - n + n * 256 = 256 + n * 255$$

Folosind 2 coduri rezervate s-ar fi putut implementa cele 696 de instrucțiuni. Pentru a simplifica în oarecare măsură electronica circuitului de comandă, care va trebui să înțeleagă instrucțiunile — decodificîndu-i codurile —, proiectanții microprocesorului **Z80** au rezervat 4 coduri. Ele sînt :

CBH, DDH, EDH, FDH

Ori de cîte ori într-o secvență de instrucțiuni apare unul din numerele de mai sus, octetul care îl urmează va fi de asemenea cod de instrucțiune, instrucțiunea fiind astfel definită de ansamblul celor două coduri.

Folosind 4 coduri de rezervă, totalul instrucțiunilor codificabile devine **1276**. Cele **1276 — 696 = 570** combinații redundante stau la originea instrucțiunilor „ascunse" ale microprocesorului **Z80**, instrucțiuni asupra cărora vom reveni în Cap. G. (COMPLEMENTE)

Din cele relatate deducem că instrucțiunile microprocesorului pot ocupa un număr variabil de octeți în memorie.

Să vedem cîți octeți se folosesc, care este deci formatul instrucțiunilor microprocesorului **Z80** !

Știm deja că **Z80** are 252 de instrucțiuni care se codifică pe un octet. Să fie deci 1 numărul octeților care trebuiesc rezervați în memoria calculatorului, pentru a descrie aceste instrucțiuni ?

Iată un exemplu :

LD H,B

Instrucțiunea înseamnă : încarcă registrul **H** cu conținutul registrului **B**. Codul instrucțiunii : 60H va conține toate informațiile necesare pentru execuția acestei instrucțiuni.

Dar ce ne facem cu următorul exemplu ?

LD H, 73_H

Această instrucțiune s-ar traduce în felul următor : încarcă registrul **H** cu valoarea 73_H, sau într-o formulare mai generală **LD r, n** unde **r** poate fi oricare registru de uz general (cu lățimea de 8 bit) și **n** un număr binar oarecare exprimat de asemenea pe un octet. Apare de la sine întrebarea : Unde se memorează octetul de date n (73_H în exemplul de sus), pentru ca microprocesorul să poată localiza ușor acest parametru, după ce a citit și înțeles codul instrucțiunii ? Soluția cea mai convenabilă este de a așeza byte-ul de date la adresa de memorie imediat următoare celei care conține octetul de cod. Această tehnică numită adresare imediată, duce la un necesar de 2 octeți pentru descrierea instrucțiunii,

dintre care unul de cod și unul de date. Conținutul celor doi octeți pentru instrucțiunea enunțată va fi deci : 26_H, 73_H.

Din prezentarea structurii interne a microprocesorului reținem faptul că regiștri simpli (1 octet) luați cîte doi, pot forma regiștri dubli (2 octeți) și că evident, există instrucțiuni care îi tratează ca atare.

Să considerăm cel mai elementar exemplu în acest sens :

LD HL, 1234_H

Încarcă registrul dublu **HL** cu valoarea hexazecimală 1234_H, sau într-o formulare generală : **LD rr, nn** unde **rr** poate fi unul din regiștrii dubli ai microprocesorului, iar **nn** orice număr binar de 16 bit (2 octeți). Ca și în cazul precedent, folosind tehnica de adresare imediată, cei doi octeți de date nn vor fi locați în memorie la adresele imediat următoare octetului de cod. Mai rămîne de lămurit doar ordinea în care se vor memora cei doi octeți. Cu o perseverență demnă de invidiat, ori de cîte ori în memorie se va scrie un cuvînt (16 bit) de date, octetul mai puțin semnificativ se va stoca la adresa inferioară, iar octetul cel mai semnificativ la adresa superioară. Amintim aici că aceeași regulă va fi respectată și de către instrucțiunea **PUSH BC** de exemplu, instrucțiune care salvează în memorie, începînd cu adresa indicată de indicatorul de stivă **SP**, conținutul registrului dublu **BC** : registrul **C** conținînd octetul mai puțin semnificativ al cuvîntului se va salva la adresa inferioară iar octetul cel mai semnificativ, din **B**, la adresa următoare. (În tehnica de calcul legile sînt făcute pentru a fi respectate !). Instrucțiunea prezentată va ocupa în memorie trei octeți, dintre care primul reprezintă codul și următorii doi datele :

21_H, 34_H, 12_H

Dacă am dori să încărcăm în aceeași manieră registrul dublu **IX**

LD IX, 1234_H

vom necesita 4 octeți :

DD_H, 21_H, 34_H, 12_H

În acest caz primii doi octeți reprezintă codul operației iar următorii doi data.

Specificînd că se pot imagina și alte combinații, sintetizăm formatul instrucțiunilor.

Instrucțiunile microprocesorului **Z80** pot ocupa în memorie 1, 2, 3 sau 4 octeți în următoarele combinații posibile :

1 octet : = 1 octet de cod		ex. : LD H,B
2 octeți : = 1 octet de cod + 1 octet de date		ex. : LD H,73_H
= 2 octeți de cod		ex. : LDI
3 octeți : = 1 octet de cod + 2 octeți de date		ex. : LD HL, 1234_H
= 2 octeți de cod + 1 octet de date		ex. : LD B, (IX+IND)
4 octeți : = 2 octeți de cod + 2 octeți de date		ex. : LD IX, 1234_H
= 3 octeți de cod + 1 octet de date		ex. : BIT 5, (IY+IND)

Putem să stabilim și cîteva **reguli** :

1. Ori de cîte ori într-o instrucțiune este implicat un registru de index **IX** sau **IY**, codul aferent instrucțiunii se va reprezenta pe doi sau trei octeți. Primul octet va fi obligatoriu DD_H pentru registrul **IX** și FD_H pentru **IY**.

2. Referirea la regiştrii index **IX** şi **IY** se face cu coduri identice cu cele pentru registrul **HL**, precedate de codul rezervat DD_H sau FD_H.

3. Instrucţiuni orientate pe bit (**BIT, SET, RES**) şi cele din clasa rotirilor şi deplasărilor cu poziţionarea indicatorilor de stare (**RLC, RRC, RL, RR, SLA, SRA, SRL**) ocupă în memorie 2 sau 4 octeţi (2 octeţi de cod respectiv 3 octeţi de cod şi unul de date). Exceptînd cazul în care ele implică regiştri de index **IX** şi **IY**, primul octet de cod este CB_H. La folosirea regiştrilor de index, codul CB_H apare — aşa cum rezultă din regula 1 — pe poziţia a doua.

4. Instrucţiunile multiple (**LDI, LDD, LDIR, LDDR, CPI, CPD, CPIR, CPDR, INI, IND, INIR, INDR, OUTI, OUTD, OTIR** şi **OTDR**) se codifică pe doi octeţi de cod, în care primul este obligatoriu ED_H.

5. Indicele **IND** utilizat pentru adresarea — folosind tehnica indexată — a unui octet din memorie, reprezintă un deplasament exprimat pe un octet faţă de o adresă de bază, conţinută în **IX** sau **IY**, iar în cîmpul instrucţiunii valoarea lui apare obligatoriu pe poziţia imediat următoare codului CB_H. Astfel apare situaţia curioasă în care într-o instrucţiune de 4 octeţi (3 cod + 1 data) cel de-al 3-lea cod (cel care specifică ceea ce este de fapt de făcut) apare pe ultima poziţie, precedată de o dată. Un exemplu în acest sens îl constituie instrucţiunea **BIT 5, (IY + IND)**, dată ca exemplu mai sus, a cărei structură este :

COD1, COD2, DATA, COD3
FDH, CBH, IND, 76H

În exemplele pe care le-am citat de-a lungul plictisitoarelor pagini de pînă aici, ne-am referit adesea la tehnici sau tipuri de adresare folosite de către microprocesor, specificînd că ele reprezintă caracteristici importante în tehnica de calcul.

Să le cunoaştem atunci !

3.2. Tehnici de adresare

„Maşina va putea să modifice conţinutul memoriei, ba mai mult, acesta este regimul normal de lucru al calculatorului. Astfel, maşina va fi capabilă să modifice instrucţiunile stocate în memorie — aceleaşi instrucţiuni care guvernează însăşi activitatea sa — putîndu-se realiza pe această cale structuri de instrucţiuni oricît de complexe, care-şi modifică în mod dinamic conţinutul, şi astfel şi procedurile de calcul pe care le are de efectuat".

Afirmaţia, care la începuturile tehnicii de calcul moderne părea puţin ciudată, dacă nu bizară de-a dreptul, s-a adeverit cu trecerea timpului.

În anii noştri există calculatoare care încearcă, din ce în ce cu mai mult succes, să-l substituie pe om în munca sa cea mai intelectuală : cea de concepţie şi cea de luare a unor decizii. Bineînţeles, că nu atît structura hardware a calculatoarelor amintite, ci mai degrabă programele implementate (software-ul) realizează aceste funcţii. Aceste pachete de software au o trăsătură esenţial nouă faţă de programele uzuale : ele au prevăzute secvenţe şi regimuri de autoinstruire. Nu putem să acceptăm ideea unui sistem expert de exemplu, care, dacă prognoza dată de el nu s-a adeverit decît în măsură de 70%, să nu fie capabil să înveţe din cele întîmplate, să se autoinstruiască. Cum vor proceda într-o asemenea situaţie programatorii, cei care au elaborat sistemul expert în cauză ? În nici un caz nu vor rescrie programele ! Ele au fost astfel concepute, încît, ataşînd prognozei calculate de ele situaţia reală, precum şi o descriere formală

a cauzelor care au condus la generarea prognozei inexacte, programele își vor recalcula automat parametrii de lucru — se vor autorestructura. Aceasta ar fi cea mai înaltă formă de materializare cunoscută pînă în prezent, a ideii neumanniene, lansată încă în anii 1940.

Nu încercăm să detaliăm structura programelor de inteligență artificială.

Am apelat la această paranteză pentru a îndrepta atenția către importanța dotării unui calculator cu cît mai multe tipuri de adresare posibile.

3.2.1. Ce sînt tehnicile de adresare ?

Așa cum am văzut, și după cum vom vedea în cele ce urmează, majoritatea instrucțiunilor lucrează cu unul sau mai mulți operanzi.

Modul în care aceşti operanzi sînt aduși la cunoștința unității centrale (microprocesorului), calea pe care i se comunică adresa la care se află locat operandul căutat, se numește tehnică de adresare.

Modul cel mai rudimentar, de a introduce un număr într-o secvență program, este de a-i specifica direct valoarea, cum ar fi și cazul instrucțiunii :

JP 5487_H

În cazul de față se va executa oricum un salt la adresa 5487_H. Tehnica pe care am folosit-o, s-ar putea numi adresare directă, fiindcă specificăm în mod direct adresa la care se va efectua saltul. Structura unui astfel de program este rigidă : saltul se va efectua mereu la aceeași adresă. Desigur, pentru a mări flexibilitatea secvenței de program care se termină cu un salt necondiționat, fără a modifica de fiecare dată conținutul instrucțiunii, ar trebui să putem specifica adresa de destinație a saltului, în mod indirect, de exemplu folosind în acest scop conținutul unui registru dublu.

Dispunem și de această posibilitate :

JP (HL)

Folosind instrucțiunea aceasta, saltul poate fi dirijat prin conținutul registrului dublu HL, conținut care reprezintă de fapt adresa de destinație a saltului program. Observăm că astfel am cîștigat în mod considerabil în flexibilitate : registrul HL poate fi implicat în operații aritmetice/logice, conținutul lui putînd fi rezultatul unor calcule oricît de complexe. Folosind această a doua formă a instrucțiunii de salt necondiționat, putem modifica secvențele de rulare a programului și implicit și efectul lui, fără a modifica conținutul (imaginea de memorie) instrucțiunii.

Instrucțiunea prezentată folosește tehnica de adresare indirectă. Specificăm că adresarea este de tip „indirect prin registru" căci există și adresare de tip „indirect prin memorie" unde adresa căutată se obține din două locații de memorie și nu dintr-un registru dublu. Acesta este cazul la acceptarea unei întreruperi în modul 2 (IM2), cînd adresa de început a rutinei de tratare a întreruperii se obține din 2 celule de memorie, care la rîndul lor nu au o adresă fixă în memorie, ci vor fi locate pe baza conținutului unor altor registre (vectorul de întreruperi provenind din registrul I și perifericul apelant). Vedem deci că în procesul de determinare a adresei de început a unei rutine de tratare a întreruperilor în mod IM2, microprocesorul recurge la o adresare dublu indirectă : indirectă prin

registru (vector) pentru determinarea adresei adresei și apoi indirectă prin memorie pentru determinarea adresei efective de salt.

Ce se întîmplă însă cu cele 2 instrucțiuni de salt prezentate, dacă dintr-un motiv oarecare vom dori să dislocăm programele în care apar instrucțiunile amintite, într-o altă zonă de memorie ? Evident, cele două instrucțiuni — dar mai ales prima — vor da greș, datorită faptului că prin dislocare s-a modificat probabil și adresa de destinație a saltului.

Peste un asemenea inconvenient se poate trece folosind o instrucțiune de salt relativ :

JR $+20_H$ de exemplu.

Efectuarea acestei instrucțiuni va genera un salt necondiționat, a cărui destinație se calculează însumînd valoarea curentă a registrului PC (contor de program) și deplasamentul specificat în enunțul instrucțiunii ($+20_H$ în cazul de față).

Este evident faptul că această instrucțiune va funcționa corect, indiferent de zona în care va fi locat programul care o conține.

Tehnica de adresare folosită este relativă și directă : relativă fiindcă operandul se specifică relativ față de o valoare curentă (variabilă) și directă, fiindcă deplasamentul se specifică explicit în cîmpul instrucțiunii. Un cîștig suplimentar de flexibilitate ar putea oferi o instrucțiune care să folosească tehnica de adresare relativă și indirectă :

jr (c) de exemplu.

Această instrucțiune ar trebui să efectueze un salt la adresa ce se obține însumînd valoarea curentă a contorului program și cea a registrului C. Din păcate setul de instrucțiuni al microprocesorului Z80 nu conține o astfel de instrucțiune. (De aceea am și notat mnemonicile instrucțiunii cu litere mici).

Dar să trecem la o prezentare metodică a tipurilor de adresare, înainte ca cititorul să rămînă cu falsa impresie că această noțiune s-ar referi doar la instrucțiunile de salt.

3.2.2. O clasificare posibilă

În literatură nu se găsește un consens general privind definirea, denumirea și clasificarea tipurilor de adresare.

De aceea, descătușați, vom defini prin analiza problemei date, tipuri de adresare posibile, invitîndu-l pe cititor la o participare efectivă și afectivă.

Să recapitulăm deci problema de rezolvat :

Trebuie să analizăm modalitățile prin care se poate specifica tot ceea ce este necesar pentru locarea unei date, pe parcursul execuției : specificarea unui operand.

Înainte de a trece la analiză reamintim faptul că, codul unei instrucțiuni trebuie să conțină și informații asupra a ceea ce are de făcut instrucțiunea respectivă, nu numai adresa datelor implicate în ea.

Pe de altă parte reținem că adresarea regiștrilor, a celulelor de memorie și a dispozitivelor de intrare/ieșire diferă esențial.

Datorită numărului în general redus al regiștrilor interni ai unui microprocesor adresele lor pot fi codificate pe cîțiva biți, pe cînd adresarea unei celule

de memorie necesită generarea unei adrese al cărei număr de biţi este egal cu numărul de linii de adresă a microprocesorului. În cazul microprocesorului **Z80** avem următorii regiştri simpli **A, B, C, D, E, H, L,** care fiind 7 la număr, adresele lor pot fi codificate pe 3 bit ($2^3 = 8$). Spaţiul de memorie adresabilă fiind $2^{16} = 65536$, pentru a adresa o celulă de memorie dată, este necesar să formăm o adresă de 16 bit. Pentru adresarea unui dispozitiv de intrare/ieşire este necesară o adresă de cel puţin 8 bit. De aici rezultă o diferenţă esenţială : codurile instrucţiunilor microprocesorului **Z80** avînd lungimea de 8 bit, pot încorpora adresa unui registru intern dar nicidecum adresa unei celule de memorie sau a unui dispozitiv de intrare/ieşire.

Şi acum să trecem la treabă :

Privind problema în modul cel mai general posibil, constatăm : o instrucţiune poate încorpora data dorită, sau adresa datei dorite. De aici încolo vă rugăm să urmăriţi arborele din fig. 3.1.

Şi cum „toate drumurile duc la Roma", tot aşa şi diversele ramuri ale arborelui nostru ; fiecare ramură finală reprezintă un tip de adresare. Structura arborelului este destul de neuniformă, iar criteriile de ramificaţie destul de eterogene, pe alocuri chiar empirice. Acest fapt nu este o întîmplare, ci este o consecinţă firească a tentativei noastre de a dirija astfel structura arborelui încît la ieşire să obţinem tipurile de adresare, aşa cum ele sînt definite şi clasificate în diversele referinţe bibliografice. Semnele de întrebare semnalează tehnici de adresare posibile, neexplorate la scară largă pînă în prezent. Ele pot reprezenta pentru cei mai ambiţioşi dintre d-voastră, puncte de reper în eventuala proiectare a unui nou tip de unitate centrală sau microprocesor.

Cei care au avut răbdarea să parcurgă cu atenţie ramurile din figura de la pag. 46 se pot considera răsplătiţi de convingerea — la care sperăm să fi ajuns — că în clasificarea tipurilor de adresare folosite actualmente, există o dezordine considerabilă.

Prezentăm pe scurt esenţa fiecăruia din cele 6 tipuri de adresare nominalizate în figură, prin prisma unor instrucţiuni ale microprocesorului **Z80.**

Adresarea imediată. Ori de cîte ori corpul unei instrucţiuni (nu codul) înglobează şi data care reprezintă obiectul acelei instrucţiuni, adresarea folosită se numeşte imediată.

Cîteva exemple în acest sens :

a) **LD A,55_H** — $3E_H$ 55_H

cod data

instrucţiune

Instrucţiunea încarcă registrul **A** cu valoarea 55_H. Data (55_H) este un octet de sine stătător, în cîmpul de doi octeţi ai instrucţiunii.

b) **LD DE,$6AF5_H$** — 11_H $F5_H$ $6A_H$

cod data

instrucţiune

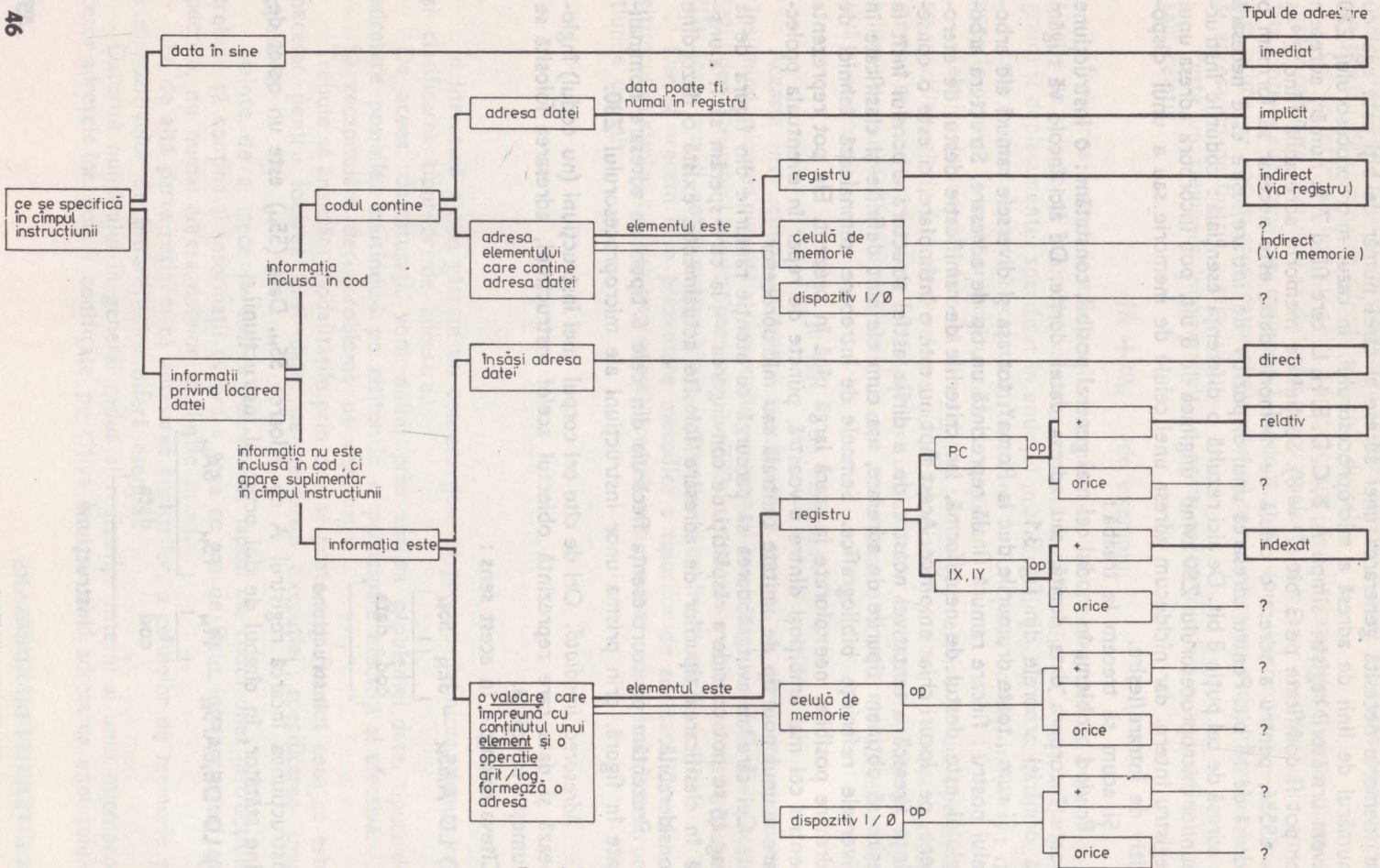
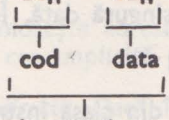


Fig. 3.1. Moduri de adresare posibile

Instrucţiunea încarcă registrul dublu DE cu valoarea hexazecimală $6AF5_H$. Data ($6AF5_H$) este un cuvînt de doi octeţi în cîmpul de 3 byte al instrucţiunii.

c) AND $8E_H$ — $E6_H$ $8E_H$



Instrucţiunea execută o operaţie logică între conţinutul registrului acumulator şi numărul $8E_H$. Data ($8E_H$) este un octet de sine stătător, în cîmpul de doi octeţi al instrucţiunii. Acest tip de adresare este cel mai rigid posibil. Un programator versat va căuta să minimizeze numărul instrucţiunilor de acest gen în lucrările sale.

Adresarea implicită. Dacă toate informaţiile necesare pentru locarea datei care reprezintă obiectul unei instrucţiuni, sînt încorporate în codul instrucţiunii, atunci adresarea se numeşte implicită.

Observăm că o dată locată într-o celulă de memorie oarecare, nu poate fi localizată prin această tehnică, deoarece adresa ei se codifică pe doi octeţi. Cu atît mai mult datele locate în regiştrii microprocesorului, a căror adresă se codifică doar pe trei bit.

Exemple de adresare implicită :

a) LD C,D — $4A_H$

Instrucţiunea transferă conţinutul registrului D în registrul C. Atît adresa iniţială a datei, cît şi adresa ei de destinaţie sînt specificate în octetul de cod. Structura byte-ului de cod este următoarea :

poziţia bitului :	b7	b6	b5	b4	b3	b2	b1	b0
valoarea :	0	1	0	0	1	0	1	0
semnificaţia :	operaţia		d2	d1	d0	s2	s1	s0

unde : **b7,b6(01)** semnifică operaţia care se execută — transfer între regiştri ;

d2,d1,d0(001) reprezintă adresa registrului de destinaţie, codificată pe trei bit ;

s2,s1,s0(010) reprezintă adresa registrului sursă, codificată pe trei bit.

Amintim adresele codificate ale regiştrilor interni :

000 — B	100 — H
001 — C	101 — L
010 — D	111 — A
011 — E	

Remarcăm deasemenea faptul că pe parcursul execuţiei acestei instrucţiuni au loc două adresări :

— una pentru citirea datei din registrul D

— a doua pentru înscrierea datei în registrul C

Observaţie : Şi în cazul exemplului a) de la adresarea imediată există o adresare implicită : adresa registrului care se încarcă (A) este încorporată în codul instrucţiunii ($3E_H$). Constatarea este valabilă şi pentru exemplele b) şi c) enunţate. De altfel marea majoritate a instrucţiunilor microprocesorului efectuează două adresări (destinaţie + sursă, sau operand 1 şi operand 2, sau destinaţie + operand). De aceea în filele de instrucţiuni, care apar în partea Complemente, vom specifica tipul de adresare folosit în ambele momente de execuţie a instrucţiunii.

b) **EI — FB_H**

Instrucțiunea validează sistemul de întreruperi al microprocesorului, înscriind valoarea 1 în bistabilele IFF_1 și IFF_2. Aceasta este una din puținele instrucțiuni care folosesc un singur tip de adresare și numai o singură dată. În cazul de față ea este bineînțeles implicită.

c) **ADD HL, BC — 09_H**

Acesta este un exemplu de adresare implicită luat din clasa instrucțiunilor aritmetice de 16 bit. Conținutul registrului dublu **HL** este adunat cu cel al registrului dublu **BC**. Și în acest caz, ambii operanzi se specifică implicit, în octetul de cod.

d) **LD DE, $6AF5_H$ — 11_H $F5_H$ $6A_H$**

Am reluat acest exemplu pentru a rememora faptul că pe parcursul execuției instrucțiunii, întîlnim atît adresarea implicită — la specificarea adresei registrului destinație, inclusă în codul 11_H — cît și adresarea imediată — folosită pentru specificarea datei ($6AF5_H$) de transferat.

Adresarea indirectă. Dacă adresa unui operand este conținutul unui registru sau a unei locații de memorie, tehnica de adresare se numește indirectă.
Adresarea indirectă „via" memorie se întîlnește într-un singur caz : atunci cînd **Z80** acceptă o întrerupere în modul 2, adresa de început a subrutinei de tratare a întreruperii este determinată folosind această tehnică.
Exemplele de adresare indirectă sînt mai numeroase. Iată cîteva :

a) **ADC (HL) — $8E_H$**

Instrucțiunea adună la conținutul acumulatorului, numărul stocat în memorie, la adresa conținută în registrul dublu **HL**, iar apoi este adăugată și valoarea bitului de transport (**carry**).
În realitate și această instrucțiune conține adresări de mai multe tipuri :

— 3 adresări implicite cuprinse în octetul de cod, pentru identificarea registrului **A**, a bitului **Cy** și a registrului de adresă **HL**.

— 1 adresare indirectă pentru citirea unui operand din memorie, operand adresat prin conținutul registrului dublu **HL**.

b) **PUSH DE — $D5_H$**

Acesta este un alt exemplu de instrucțiune, care folosește adresarea indirectă. Efectul ei este : conținutul regiștrilor **D** și **E** este salvat în memorie la adresa următoare din stivă, începînd cu registrul **D**. Adresa la care se face transferul este conținută în indicatorul de stivă **SP**, al cărui conținut este decrementat după fiecare transfer.
Instrucțiunea folosește următoarele tehnici de adresare :

— implicit : — de 3 ori ; 2 pentru a specifica cei doi regiștri **D** și **E**, de unde se ia data și 1 pentru a specifica registrul de adresă **SP**
— indirect : — adresa de destinație a transferului în memorie se ia din registrul **SP**.

c) **LD A,(BC) — $0A_H$**

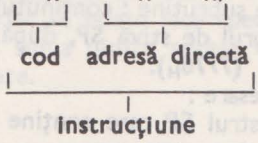
Instrucțiunea încarcă conținutul registrului **A**, din memorie, de la adresa conținută în registrul dublu **BC**.

Instrucțiunea folosește atît adresarea implicită (pentru specificarea destinației — registrul **A**) cît și adresarea indirectă (pentru a specifica adresa sursei — celula de memorie adresată prin conținutul registrului dublu **BC**).

d) **RST 08H — CF_H**
Instrucțiunea efectuează un salt necondiționat la adresa **0008H**, după ce în prealabil salvase pe stivă (la adresa conținută în registrul **SP**) valoarea contorului de program **PC**.
Instrucțiunea folosește adresarea indirectă (pentru salvarea în memorie a conținutului **PC**) și cea implicită pentru a efectua saltul.

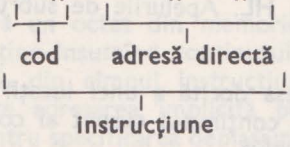
Adresarea directă. Dacă în corpul unei instrucțiuni apare o adresă, adresă care este adresa efectivă a datei ce constituie obiectul instrucțiunii, atunci adresarea se numește directă.
Adresarea directă apare mai ales la instrucțiunile de transfer între regiștri și memorie, precum și la instrucțiunile de salt.
Folosirea instrucțiunilor ce utilizează acest tip de adresare, se va limita la un minim posibil, deoarece ele rigidizează programul.
Pentru a putea rula în altă zonă de memorie decît cea originală, instrucțiunile din program, ce folosesc adresarea directă, vor trebui probabil modificate.
Exemple :

a) **LD (1784_H),A** — 32_H 84_H 17_H

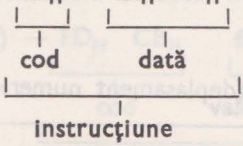


Această instrucțiune transferă conținutul registrului **A** în memorie, la adresa 1784_H.
Adresarea folosită este implicită pentru determinarea sursei (reg. **A**) și directă pentru specificarea destinației (adresa 1784_H).

b) **LD HL, (1122_H)** — $2A_H$ 22_H 11_H



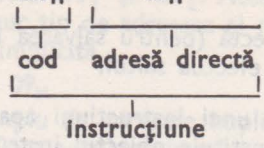
LD HL, 1122_H — 21_H 22_H 11_H



Am enunțat alăturat cele două instrucțiuni pentru a atrage atenția asupra pericolului de confuzie, care apare la aceste două instrucțiuni. Întrucît ambele cîmpuri de instrucțiuni conțin un cod pe un octet și o valoare numerică pe doi octeți, am putea fi tentați să considerăm ambele cazuri, ca exemple de adresare imediată. Dar nu este așa. În prima instrucțiune 1122_H este adresa datei care constituie obiectul instrucțiunii, pe cînd în cel de-al doilea exemplu 1122_H reprezintă însăși data cu care operează instrucțiunea.
LD HL, (1122_H) — conține și o *adresare directă* ;
LD HL, 1122_H — conține și o *adresare imediată* ;

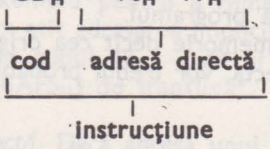
ambele conţin adresarea implicită pentru specificarea destinaţiei : registrul dublu **HL**.

c) **OUT (80H),A** — $D3_H$ 80_H



Instrucţiunea transferă conţinutul registrului acumulator **A** la dispozitivul de ieşire care are adresa 80_H. Adresa sursei **(A)** este specificată implicit. iar cea a destinaţiei (portul cu adresa 80_H) direct.

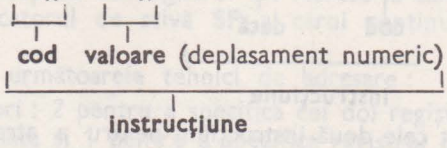
d) **CALL** 7778_H — CD_H 78_H 77_H



Instrucţiunea reprezintă un apel tipic de subrutine : conţinutul contorului program este salvat în stiva adresată de indicatorul de stivă **SP**, după care se execută un salt la adresa de început a subrutinei (7778_H).
Instrucţiunea foloseşte 3 tipuri de adresare :
— **implicită** — pentru a specifica registrul **SP** care conţine adresa la care se va face salvarea în memorie
— **indirectă** — pentru a specifica adresa de memorie în care se va salva **PC** (conţinutul registrului **SP**)
— **directă** — pentru a specifica adresa la care se va efectua saltul.
Păcat că proiectanţii microprocesorului **Z80** n-au implementat şi un apel de subrutină, la care adresa de salt să poată fi specificată şi ea indirect. De exemplu prin conţinutul registrului dublu **HL**. Apelurile de subrutină ar fi fost în acest caz mult mai flexibile.

Adresarea relativă. Dacă adresa dorită a unei locaţii de memorie se obţine adăugînd o valoare (oarecare) la conţinutul curent al contorului program **(PC)**, atunci adresarea se numeşte relativă.

Exemplu : **JR** $+05_H$ — 18_H 03_H



Instrucţiunea execută un salt la adresa care se calculează însumînd valoarea curentă a contorului program PC cu valoarea (deplasamentul) inclusă în cîmpul instrucţiunii. Nu trebuie să ne deranjeze faptul că în enunţul instrucţiunii deplasamentul are valoarea $+05_H$ iar cîmpul cod $+03_H$. Diferenţa de 2 apare datorită faptului că dacă instrucţiunea este locată la adresa 100_H de exemplu, atunci în momentul adunării deplasamentului, PC conţine deja 102_H. Astfel se vine în ajutorul programatorului care vede pe listing adresa de început a instrucţiunii. Transpunerea deplasamentului este efectuată automat de orice asamblor cît de cît „cinstit".

Ce salturi relative frumoase s-ar fi putut executa dacă valoarea deplasamentului s-ar fi putut specifica nu numai imediat (incluzînd-o în corpul instrucțiunii) ci și indirect (de ex. prin valoarea conținută în registrul C) !
Din păcate **Z80** nu cunoaște această posibilitate.
Constatăm deci că instrucțiunea prezentată utilizează adresarea imediată pentru specificarea deplasamentului și adresarea relativă pentru determinarea adresei de salt.
Dacă un program conține numai salturi relative (și nu utilizează adresarea directă) atunci el poate rula la orice adresă de memorie, fără modificări.

Adresarea indexată. Dacă adresa unei date, ce constituie obiectul unei instrucțiuni, se obține adăugînd la un registru de bază (index), un deplasament (indice) atunci adresarea se numește indexată.
Z80 posedă 2 regiștri de 16 bit **IX** și **IY**, care pot fi folosiți ca regiștri de bază. Tehnica de adresare indexată este eficientă în cazul în care dispunem de date organizate într-un tabel, aidoma unei variabile cu indici. Regăsirea variabilelor cu indici trebuie să se poată face pur și simplu specificîndu-le indicele.
Considerînd că fiecare element din tabel ocupă un octet și că adresa de început (adresa de bază) a tabelului se încarcă în registrul index **IX** sau **IY**, atunci regăsirea unei date dorite se poate face specificîndu-i doar indicele. Microprocesorul va realiza automat suma dintre adresa de bază și indice, obținînd astfel adresa fizică a datei căutate.
Exemple :

a) **LD E, (IX + 25_H)** — **DD_H $5E_H$ 25_H**

cod **valoare (indice)**

instrucțiune

Instrucțiunea transferă un octet din memorie în registrul E. Adresa celulei de memorie sursă se obține însumînd conținutul registrului de bază **IX** și valoarea deplasamentului **+25_H** din cîmpul instrucțiunii.
Instrucțiunea folosește adresarea implicită pentru a specifica regiștrii **E** și **IX**, adresarea imediată pentru specificarea deplasamentului și adresarea indexată pentru obținerea datei ce constituie obiectul instrucțiunii.

b) **RES 7, (IY + 69_H)** — **FD_H CB_H 69_H BE_H**

cod **valoare (indice)** **cod**

instrucțiune

Instrucțiunea șterge (înscrie **0**) bitul cel mai semnificativ al octetului din memorie, octet a cărui adresă se obține însumînd conținutul registrului de bază (index) **IY** și valoarea numerică (indice) **69_H** conținută în cîmpul instrucțiunii. Observăm că instrucțiunea ocupă 4 octeți din care trei octeți sînt rezervați pentru cod.
Instrucțiunea folosește următoarele tehnici de adresare :
— adresare implicită : pentru a specifica bitul afectat (7) și registrul de bază ce va fi folosit (**IY**)

— adresare imediată : pentru a specifica valoarea deplasamentului, valoare inclusă în corpul instrucțiunii ;
— adresare indexată : pentru a calcula adresa fizică a octetului ce va fi manipulat (însumînd conținutul registrului de bază **IY** și deplasamentul 69_H)

Păcat că nu avem posibilitatea de a specifica deplasamentul în mod indirect (ex. conținutul registrului **D**).

Atîta timp, cît vom fi obligați să includem indicele numărului dorit în mod explicit în cîmpul instrucțiunii, va fi destul de greu să baleiem în mod selectiv o tabelă de exemplu.

Facilitatea amintită nu este prevăzută în „logica cablată" a microprocesorului **Z80.** Ar fi totuși o posibilitate de a varia în mod dinamic, prin program, valoarea unui indice.

Și anume : cunoscînd exact adresa locației de memorie care conține indicele situat în cîmpul instrucțiunii, putem concepe un program care să incrementeze sau să decrementeze, conform regulei dorite, conținutul acelei celule de memorie și astfel și indicele. Dar această metodă nu o recomandăm nici dușmanilor noștri. Din 2 motive :

a) un asemenea program „moare subit" dacă este transpus într-o memorie fixă (de tip **ROM**) ;

b) urmărirea, depanarea, service-ul unui program care își schimbă mereu instrucțiunile este deosebit de greoaie.

Din prezentarea tipurilor de adresare relativă și indexată (ca și din fig. 3.1 de altfel) rezultă că cele două tipuri sînt aproape identice din punctul de vedere al mecanismelor pe care le invocă. Discriminarea celor 2 tehnici de adresare s-a făcut în cazul de față nu pe baza unor diferențe de ordin tehnic, ci mai degrabă pe baza noțiunilor de programare ce le-au fost asociate. Remarcăm deasemenea că în esență ele reprezintă un tip de adresare indirectă („via" registru) modificat, în care adresa de memorie nu se obține nemijlocit dintr-un registru ci adunîndu-i o constantă numerică. Credem că fenomenul poate fi folosit ca un etalon de clasificare neriguroasă. Ea are rădăcini ce țin de evoluția și istoria tehnicii de calcul. Am păstrat-o pentru a nu intra în flagrantă contradicție cu alte referințe bibliografice, scutindu-l astfel pe cititorul novice de eventuale clipe de descumpănire.

Există în schimb unele tipuri de adresare definite în documentația de firmă [MOSTEK] care nu le-am acceptat.

3.2.3. Critica documentației de firmă

În [30] p. 20—22 se enunță 10 tipuri de adresare. Le vom aminti pe cele pe care noi le-am respins, asociindu-le la una din cele 6 tipuri de adresare prezentate mai sus.

2. *Immediate Extended Addressing (imediat extins).* În documentația de firmă instrucțiunile de tipul **LD H, 12_H** și **LD HL, 1234_H** sînt tratate ca instrucțiuni ce folosesc tipuri diferite de adresare :

LD H, 12_H — imediat
LD HL, 1234_H — imediat extins

Noi considerăm că în esență ambele instrucțiuni folosesc adresarea imediată. Discriminarea lor este artificială și forțată.

3. Modified Page Zero Addressing (pagina zero modificată)
În această clasă sînt introduse instrucțiunile **RST** (restart) pe care noi le considerăm exemplul etalon de adresare implicită. Informația referitoare la adresa de salt a acestor instrucțiuni este introdusă sub formă codificată în însuși codul instrucțiunii. Faptul că salturile se efectuează în pagina **0** a memoriei **(A0—A15** fiind cuprinse în limitele **(0,255))**, este un amănunt nesemnificativ din punctul de vedere al tehnicii de adresare folosite.

7. Register Addressing (adresare de tip registru)
Toate instrucțiunile microprocesorului **Z80** care implică numai regiștri interni, folosesc tehnica de adresare implicită în care informațiile necesare pentru locarea operanzilor se încorporează în însuși codul instrucțiunii. Considerăm introducerea acestui tip de adresare ca nejustificată.

10. Bit Addressing (adresare de tip bit).
Instrucțiunile orientate pe bit **(BIT, SET, RES)** specifică numărul bitului afectat în însuși codul operației. De aceea noi considerăm că ele folosesc adresarea implicită. Introducerea tehnicii de adresare tip bit, o considerăm inutilă.
Tehnica de adresare pe care noi am numit-o directă, apare sub denumirea Extended Addressing. Problema numelui acordată unui fenomen fiind o chestiune de gust, acceptăm și această variantă.

Credem că argumentele pe care le-am invocat pentru a valida deciziile luate, exclud orice dubiu privind justețea lor.

Căutînd totuși o explicație pentru această clasificare insuficient de riguroasă nu găsim altă motivare decît cea pe care o etalăm în continuare.

În primul rînd, sîntem convinși că această clasificare s-a preluat din prima documentație a microprocesorului **Z80**, publicată de firma Zilog în 1975, păstrîndu-se nemodificată poate și datorită faptului că terenul tipurilor de adresare este destul de mlăștinos, datorită în primul rînd criteriilor de clasificare incoerente încetățenite de-a lungul anilor.

Dar de ce au fost publicate, așa cum au fost, la prima publicare a documentației **Z80**?

Este cert că cei care au lansat pe piață microprocesorul **Z80** au dorit să aducă cît mai multe argumente în favoarea „odraslei" lor, sărind puțin peste cal. Zece tipuri de adresare înseamnă mult, foarte mult. Oare totalul instrucțiunilor 696, față de cele 244 ale marelui rival **(I8080)**, n-ar fi ajuns?!

După ce i-am făcut cu ou și cu oțet pe cei de la Zilog, revenim la problemele noastre, trăgînd concluzia că o unitate centrală (microprocesor) este considerată a fi cu atît mai puternică, cu cît numărul tehnicilor de adresare pe care le cunoaște este mai mare și cu cît acestea sînt mai complexe — adresa *(adresei (adresei (...))*.

3.3. Codul mașină și limbajul de asamblare

21_H 00_H 00_H 23_H $7D_H$ $B4_H$ $C2_H$ 03_H 01_H $C9_H$

Șirul numerelor de mai sus nu este altceva decît un program scris în cod mașină. El efectuează cea mai lungă temporizare ce se poate realiza prin decrementarea succesivă a unui registru dublu de 16 bit. Această secvență de cod

poate fi executată direct atît cu microprocesorul **I8080** cît și cu Z80 și este un exemplu de programare în cod mașină.

Cu zece ani în urmă nu puțini ar fi fost acei colegi de-ai noștri care ar fi înțeles fără nici un fel de dificultate șirul codurilor mai sus enunțate. Nu pentru că ar fi avut o plăcere deosebită în a memora numere binare exprimate hexazecimal, ci datorită faptului că orice început de drum este greu, pionierii dintotdeauna fiind adesea obligați să-și irosească forțele tăind cărări printre stînci și construind cetăți de nisip pe terenuri mlăștinoase. Cam așa se scriau codurile direct executabile la începuturile calculatoarelor electronice. Continuînd pe acest făgaș programatorii microprocesorului **I8080** ar fi fost obligați să rețină „doar" 244 de coduri de 1 octet. Cei care și-ar fi asumat riscul de a programa microprocesorul **Z80** ar fi vehiculat 696 de coduri de 1, 2 sau 3 octeți. Din fericire intelectul uman n-a tolerat multă vreme această sclavie, apropiind mai întîi codurile mașină de gîndirea și vorbirea curentă prin elaborarea limbajelor de asamblare și creînd apoi programe de translatare automată a instrucțiunilor, descrise în acest limbaj, în cod mașină.

Codurile instrucțiunilor se formulează în limbajul de asamblare ca și cuvinte cheie — prescurtate de obicei — care exprimă funcția reprezentată de codul respectiv. Aceste cuvinte cheie se numesc **mnemonici.**

Datele incluse în cîmpul instrucțiunii cod mașină, și cele pe care instrucțiunile le invocă, apar în limbajul de asamblare ca și **operanzi** atașați mnemonicei. S-a creat astfel un *mediu* — **limbajul de asamblare** — în care libertatea de gîndire (a programatorului) a crescut considerabil, și o *unealtă* — **programul de translatare,** numit **asamblor** — care a sporit fără îndoială eficiența muncii de programare.

Folosind mnemonicele microprocesorului **I8080** programul cod mașină citat, se scrie :

```
(*)   LXI   H,0000H
      INX   H
      MOV   A,L
      ORA   H
      JNZ   0103H ; (*)
      RET
```

Mnemonicile microprocesorului **Z80** exprimă aceeași realitate sub o formă puțin diferită :

```
(*)   LD    HL,0000H
      INC   HL
      LD    A,L
      OR    H
      JP    NZ,0103H ; (*)
      RET
```

Povestea programului : încarcă registrul dublu **HL** cu **0** ; incrementează conținutul registrului **HL** cu 1 ; vezi dacă **HL**=0 ; dacă nu, atunci reia operațiile de la incrementare (HL crește de la 0000_H pînă la $FFFF_H$, urmat de 0000_H) ; dacă nu, atunci gata.

Remarcăm că fiecare microprocesor are limbajul său propriu de asamblare.

Reținem că limbajul de asamblare este o imagine fidelă a setului de instrucțiuni, el exprimînd cît se poate de exact operațiile elementare pe care o unitate centrală de calculator (sau un microprocesor) le poate executa.

Între instrucțiunile limbajului de asamblare și codurile binare de instrucțiuni ale unui calculator există totdeauna o relație biunivocă : fiecărei instrucțiuni în limbaj de asamblare îi corespunde unul și numai un cod.

Forma generală a unei instrucțiuni în limbajele de asamblare ale microprocesoarelor **I8080** și **Z80** este :

Mnemonică Operand 1, Operand 2

Operanzii sînt opționali : poate apare doar unul, sau pot lipsi.

Exemple :

2 operanzi	**MOV A,L**	**LD A,L**
1 operand	**INX H**	**INC HL**
0 operanzi	**RET**	**RET**

Mnemonica este de obicei un nume generic și poate descrie mai multe instrucțiuni. În acest caz, instrucțiunile vor fi discriminate pe baza operanzilor atașați mnemonicei.

Vom încerca în paragraful următor și în cap. E (COMPLEMENTE) să ordonăm în diverse clase și grupe funcționale cele 696 de instrucțiuni ale microprocesorului Z80.

Cei nerăbdători pot răsfoi de pe acuma paginile cap. E pentru a-și forma o imagine de ansamblu asupra setului de instrucțiuni ai microprocesorului Z80, și asupra mnemonicelor utilizate în limbajul de asamblare al acestui microprocesor.

3.4. Clasificarea instrucțiunilor

Pe baza funcției lor declarate, am împărțit cele 696 de instrucțiuni ale microprocesorului Z80 în 12 clase distincte.*

Claselor le-am acordat cîte un nume simbolic.

În cadrul claselor am efectuat o subîmpărțire pe grupe de instrucțiuni.

Redăm în continuare prezentarea condensată a claselor alese :

Clasa 1.	**Instrucțiuni de transfer de 8 bit**	**LOAD—8**
(p. 278)	Mnemonici incluse : LD	
	Totalul instrucțiunilor din clasă : 111	
	Numărul de grupe : 11	
Clasa 2.	**Instrucțiuni de transfer de 16 bit**	**LOAD—16**
(p. 289)	Mnemonici incluse : LD,EX,EXX,POP,PUSH	
	Totalul instrucțiunilor din clasă : 39	
	Numărul de grupe : 10	
Clasa 3.	**Instrucțiuni de transfer de blocuri de date**	**LOAD—IDR**
(p. 301)	Mnemonici incluse : LDD,LDDR,LDI,LDIR	
	Totalul instrucțiunilor din clasă : 4	
	Numărul de grupe : 2	

* Clasificarea aparține autorului

Clasa 4. **Instrucţiuni aritmetice/logice pe 8 bit** AR/LOG—8
(p. 303) Mnemonici incluse : ADD,ADC,SUB,SBC,AND,XOR,OR,CP,INC,
DEC,CPL,NEG,DAA,RLCA,RRCA,RLA,RRA
Totalul instrucţiunilor din clasă : 115
Numărul de grupe : 26

Clasa 5. **Instrucţiuni aritmetice de 16 bit** ARIT—16
(p. 336) Mnemonici incluse : ADD,ADC,SBC,INC,DEC
Totalul instrucţiunilor din clasă : 32
Numărul de grupe : 7

Clasa 6. **Instrucţiuni logice pe blocuri de date** LOG—IDR
(p. 343) Mnemonici incluse : CPD,CPDR,CPI,CPIR
Totalul instrucţiunilor din clasă : 4
Numărul de grupe : 2

Clasa 7. **Instrucţiuni orientate pe bit** BiTSR
(p. 346) Mnemonici incluse : BIT,SET,RES,SCF,CCF
Totalul instrucţiunilor din clase : 242
Numărul de grupe : 7

Clasa 8. **Instrucţiuni de salt** JUMP
(p. 353) Mnemonici incluse : JP,JR,DJNZ
Totalul instrucţiunilor din clasă : 18
Numărul de grupe : 6

Clasa 9. **Instrucţiuni de apel şi revenire din subrutină** CALL/RET
(p. 359) Mnemonici incluse : CALL,RET,RST
Totalul instrucţiunilor din clasă : 26
Numărul de grupe : 5

Clasa 10. **Instrucţiuni de rotire şi deplasare** ROT/SHIFT
(p. 366) Mnemonici incluse : RLC,RRC,RL,RR,SLA,SRA,SRL,RLD,RRD
Totalul instrucţiunilor din clasă : 72
Numărul de grupe : 9

Clasa 11. **Instrucţiuni de intrare/ieşire** IN/OUT
(p. 381) Mnemonici incluse : IN,IND,INDR,INI,INIR,OTDR,OTIR,OUT,OUTD
OUTI
Totalul instrucţiunilor din clasă : 24
Numărul de grupe : 8

Clasa 12. **Instrucţiuni de comandă** SYS
(p. 389) Mnemonici incluse : DI,EI,HALT,IM,NOP,RETI,RETN
Totalul instrucţiunilor din clasă : 9
Numărul de grupe : 5

În dreptul fiecărei clase am indicat paginile pe care se găseşte descrierea detaliată a instrucţiunilor aferente.

Cei nerăbdători pot răsfoi de pe acum paginile respective precum şi cele premergătoare lor (§ E.1) unde se regăseşte şi subîmpărţirea pe grupe a instrucţiunilor din fiecare clasă tratată.

În măsura în care numele pe care le-am ales pentru specificarea claselor şi a paginilor au fost suficient de sugestive, cititorul îşi poate forma deja o impresie asupra setului de instrucţiuni şi implicit despre posibilităţile microprocesorului Z80.

4

CICLURI DE BAZĂ

Microprocesorul este, din punctul de vedere al modului de eşalonare în timp a activităţii sale, un dispozitiv sincron : toate manevrele sale interne şi externe se întîmplă sincron (deodată) cu fronturile semnalului de tact (clock) care excită structura, „via" pinul nr. 6 al capsulei. *Privit de sus, de foarte sus, de unde totul pare ca un vis simplu şi frumos, microprocesorul se arată a fi un element de rînd, a cărui activitate se poate reduce la două momente :*

a) citeşte o instrucţiune din memorie

b) execută instrucţiunea citită

După care totul începe de la început şi se repetă. Aparent, monotonia acestei vieţi este totală, amplificată parcă de adierea uniformă şi zumzetul invariabil al ventilatorului menit să prelungească la nesfîrşit o viaţă în care puţinele raze de soare se infiltrează prin gratiile fantelor de răcire, şi sursele de alimentare menţin tensiunea constantă. Dacă ar fi s-o vedem şi s-o auzim (VISIBLE—Z80 ne-a permis acest lucru) impresia noastră ar întări afirmaţia de sus. Un singur element pare suspect : vehemenţa mişcărilor din capsulă. Cei mărunţi dau din mîini şi... le mişcă repede. De trei patru milioane de ori pe secundă! Să ne fi înşelat oare? Să fie ceea ce vedem, nu monotonie, ci extaz continuu? Să fie oare deşertul doar o aparenţă, un voal după care se ascunde o viaţă exuberantă? Asemenea optimistului înverşunat care nu se împotmoleşte niciodată, vom încerca să descoperim împreună frumuseţea amănuntelor, pentru a vă trezi pofta de ... citire.

4.1. Ciclul de tact, ciclul maşină, ciclul instrucţiune

Am mai amintit, dar reluăm ideea că cea mai simplă instrucţiune se execută în 4 perioade ale semnalului de tact, iar cea mai lungă în 23. Să luăm cîte un exemplu din cele două clase :

4 taci : SCF — codul : 37_H

23 taci : SET 0 (IX+59_H) — codul : $DD_HCB_H59_HC6_H$

Cele două instrucţiuni par a fi înrudite şi totuşi ce deosebire ! ambele înscriu valoarea 1 în cîte un bit al unui octet : SCF (Set Carry Flag — înscrie indica-

torul de transport) poziționează la 1 bitul cel mai puțin semnificativ al registrului F, iar **SET 0, (IX+IND)** are același efect dar asupra unui octet locat în memorie la adresa care se obține însumînd conținutul registrului dublu **IX** și valoarea 59_H. Cititorul atent a remarcat faptul că prima instrucțiune este descrisă printr-un cod de 1 octet (37_H) pe cînd cea de a doua prin 4 octeți. Trecem la definiții

Definiția 1. Entitatea care însumează toate activitățile care se desfășoară în timpul unei perioade a semnalului de ceas (tact) se numește **ciclu sau stare de tact. (CC—Clock Cycle).**

Definiția 2. Entitatea care reunește toate activitățile care se desfășoară pe parcursul execuției unei instrucțiuni a cărei cod este locat în memorie, se numește **ciclu instrucțiune. (IC-Instruction Cycle).**

Pe cînd durata unui ciclu de instrucțiune variază de la o clasă la alta, durata ciclului de tact este unică pentru un microcalculator dat, perioada T a semnalului de tact fiind unitatea de măsură a timpului elementar consumat de procesora Analizînd modul de execuție a celei de a doua instrucțiuni (SET 0, ($IX+59_H$). observăm că cei 23 de tacți se compun din mai multe activități pe care în prim) aproximație le-am putea grupa astfel :

1. Se efectuează 4 citiri din memorie pentru citirea instrucțiunii **(DD_H $C3_H$ 59_H $C6_H$).**
2. Se efectuează o citire din memorie de la adresa **$IX + 59_H$**
3. Se rescrie la această adresă, octetul în care deja **$b_0 = 0$.**

Observăm că pentru a efectua instrucțiunea analizată microprocesorul va realiza 6 accese la memorie, din care 5 sînt de tip citire iar unul de tip scriere. Oricine își poate imagina că cele 5 activități de citire a memoriei sînt oricum activități înrudite. Este deci foarte probabil că în cursul efectuării instrucțiunii alese ca exemplu, microprocesorul va repeta de cîteva ori anumite secvențe de lucru.

În plus, asemenea activități se regăsesc la execuția majorității instrucțiun oricărei unități centrale. Pentru a conferi un plus de modularitate și flexibilita în execuția instrucțiunilor precum și pentru a ușura implementarea hardware-ul din chip, ciclul instrucțiune a fost subdivizat în cîteva tipuri de activități procesor intermediare.

Definiția 3. Suma activităților procesor care rezolvă o activitate intermediară bine definită, avînd o finalitate clară, în cursul execuției une instrucțiuni, se numește **ciclu mașină. (MC-Machine Cycle).**

Considerentele care au dus la subdivizarea ciclurilor instrucțiune în cicluri mașină au fost de ordin tehnic, dar ele pot fi invocate și ca argumentare filozofică : infinitul nu poate fi cuprins (exprimat) printr-o clasă infinită de elemente, scop deplin realizabil însă dacă se folosește o clasă cu un număr limitat de elemente judicios alese. Infinitatea tonalităților limbilor indo-europene pot fi formate și exprimate prin seturi de litere (vocale și consoane) care derivă din cele doar 26 litere ale alfabetului roman. Scrisul chinezesc însă, cel care folosește figuri descriptive în loc de litere, posedă o clasă de cîteva mii de elemente grafice, clasă care practic nu se va închide niciodată. Într-un mod similar : în loc de a realiza un circuit de comandă care să rețină 696 de secvențe distincte de acti-

vităţi, menite să implementeze setul de instrucţiuni al microprocesorului nostru, s-a recurs la stabilirea unor cicluri maşină puţine la număr, cu ajutorul cărora realizarea circuitului de comandă al microprocesorului s-a simplificat considerabil. Care ar fi aceste activităţi tipizate (cicluri maşină) ?

● **Citirea şi decodificarea codului de operaţie (fetch)**

Ştim deja că o instrucţiune constă din 1—4 octeţi, din care unul reprezintă obligatoriu codul de instrucţiune. Abia după citirea şi înţelegerea (decodificarea) acestui cod, microprocesorul va şti ce are de făcut pentru a executa instrucţiunea.

● **Citirea unui octet de date (memory read).**

Datele numerice care apar în cîmpul unei instrucţiuni se citesc din memorie şi se depun într-unul din regiştri interni ai microprocesorului. Operaţia fiind mai simplă şi deci mai scurtă în timp, decît cea precedentă a dus la realizarea a două tipuri de cicluri maşină de citire din memorie :

— ciclu de citire a codului de operaţie (fetch)

— ciclu de citire a datelor numerice (read)

● **Scrierea unui octet în memorie (memory write).**

Aşa cum am remarcat şi în exemplul analizat, anumite instrucţiuni vor înscrie date numerice în memorie. Ciclul maşină care rezolvă operaţia se numeşte ciclu maşină de scriere în memorie : write.

Folosind aceste trei cicluri maşină, marea majoritate a instrucţiunilor microprocesorului Z80 pot fi implementate. Există şi cicluri speciale în activitatea procesorului, dar ele vor fi tratate în cap. 8 şi cap. 9

Sintetizăm funcţionarea microprocesorului afirmînd :

— instrucţiunile se execută în cicluri de instrucţiune (IC) ;

— ciclurile instrucţiune sînt constituite dintr-un număr întreg de cicluri maşină (MC) ;

— Ciclurile maşină sînt constituite dintr-un număr întreg de cicluri de tact (CC).

Ciclurile maşină ale microprocesorului Z80 sînt :

a) ciclul de citire al codului de operaţie **(fetch) (M1)**
b) ciclul de citire al unui octet din memorie **(read)**
c) ciclul de scriere al unui octet în memorie **(write)**
d) ciclul de citire a unui dispozitiv de intrare **(in)**
e) ciclul de scriere a unui dispozitiv de ieşire **(out)**
f) ciclul intern
g) ciclul de acceptare a cererii de întrerupere
h) ciclul de acceptare a cererii de magistrale

Frecvenţa de apariţie a tipurilor **a** — **c** este mult mai mare decît a celorlalte. De aceea le numim **cicluri de bază.** Le vom trata în prezentul capitol.

În fig. 4.1 am reprezentat ciclurile de bază, folosind ca unitate etalon ciclul de tact **T.**

Vom recurge în sfîrşit la **VISIBLE—Z80,** pentru a ilustra activităţile întreprinse de procesor în vederea efectuării instrucţiunilor. Aşa cum se poate vedea în oricare dintre imaginile create cu **VISIBLE—Z80,** în partea din stînga sus a ecra-

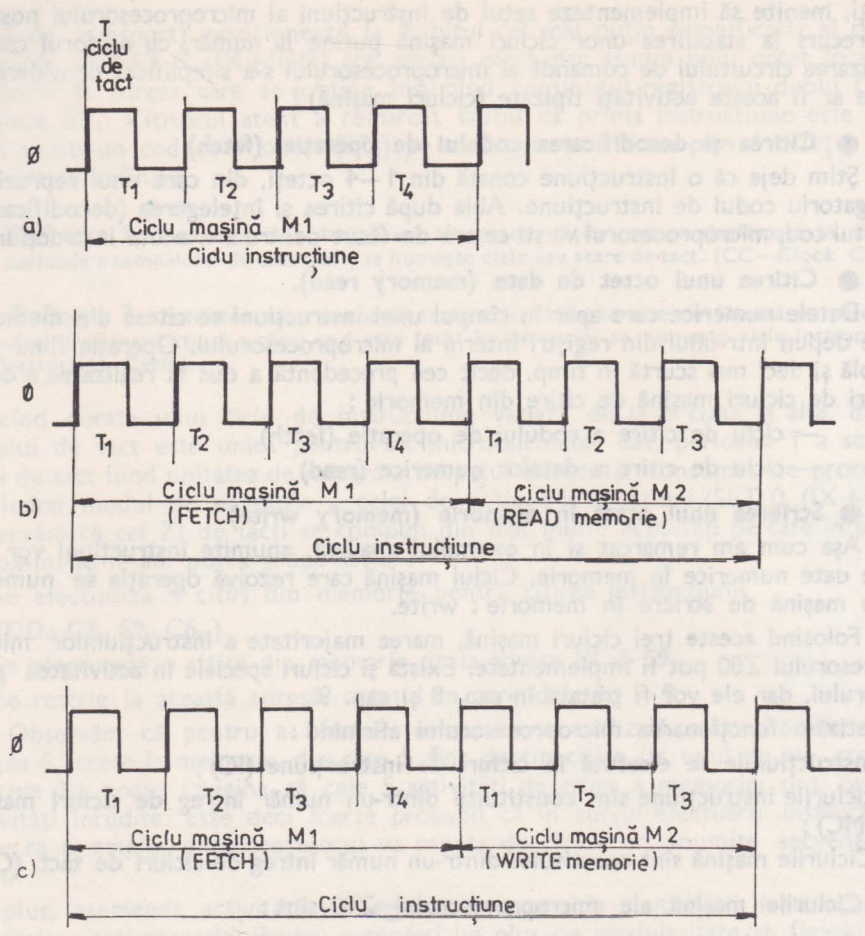


Fig. 4.1. Cicluri mașină de bază

nului, am rezervat un cîmp de comentarii în care indicăm adresa de locare a instrucțiunii în memorie (ADR), mnemonica instrucțiunii (MNE), codul ei (COD), precum și ciclul de tact (T) și cel de mașină (CM), curent.

4.2. Ciclul mașină M1 (FETCH)

Primul ciclu mașină (M1) efectuat pe parcursul unei instrucțiuni, are un rol aparte. Pe parcursul lui se efectuează citirea și interpretarea codului instrucțiunii. Microprocesorul posedă un pin dedicat, pe care generează semnalul $\overline{M1}$. Activarea acestui semnal, aduce la cunoștința lumii înconjurătoare faptul că microprocesorul execută cel mai important ciclu mașină.

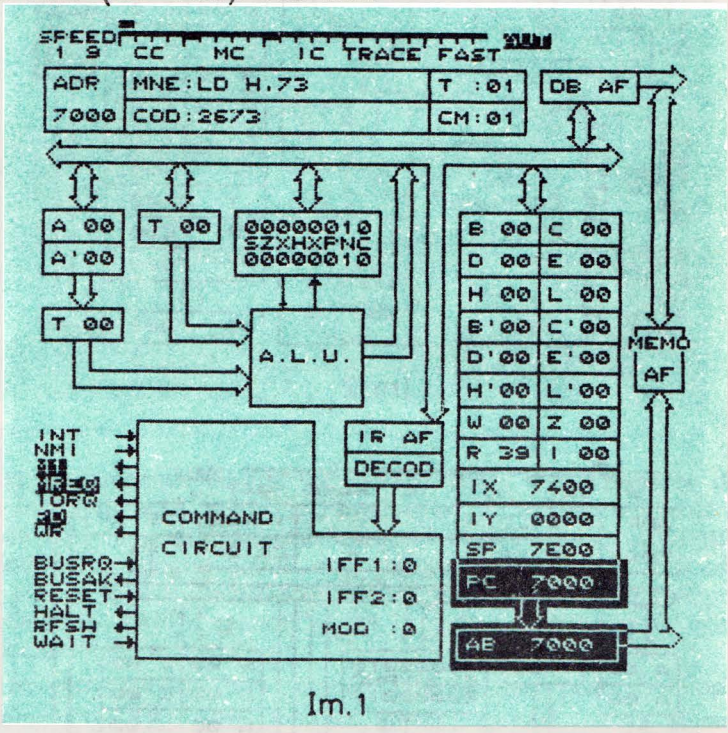
Să considerăm instrucțiunea :

LD H,73_H

Codul instrucțiunii este : 26_H 73_H

Semnificația ei : încarcă registrul **H** cu 73_H.

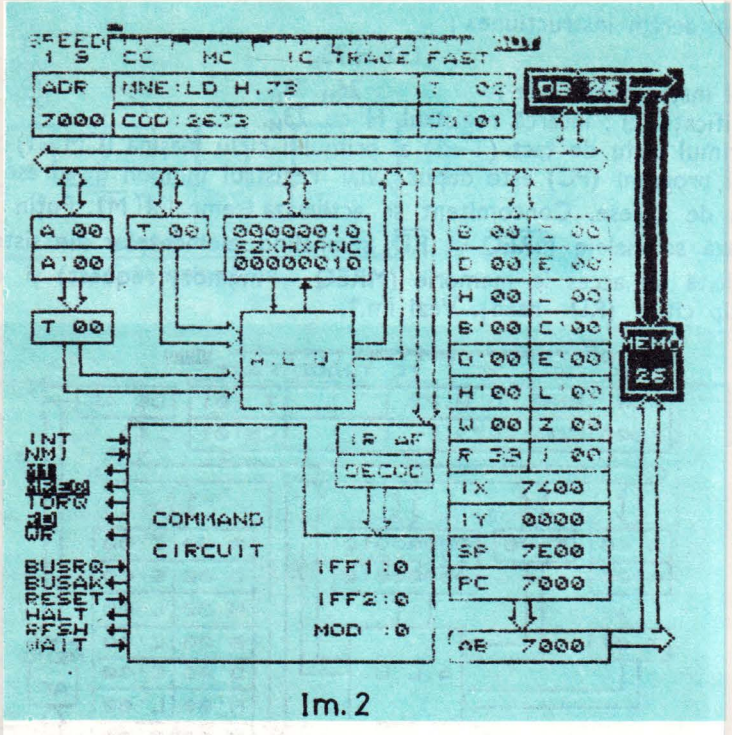
În primul ciclu de tact (T=1) al primului ciclu mașină (CM=1) conținutul contorului program (**PC**) este depus „via" registrul tampon de adrese (**AB**) pe magistrala de adrese. Concomitent se activează semnalul $\overline{M1}$. Puțin mai tîrziu se activează semnalele $\overline{MREQ}$ și $\overline{RD}$, semnalînd elementelor din sistem faptul că se solicită un acces la memorie (**MREQ — memory request**) și că accesul este de tip citire (**RD—read**). Vezi Im.1.



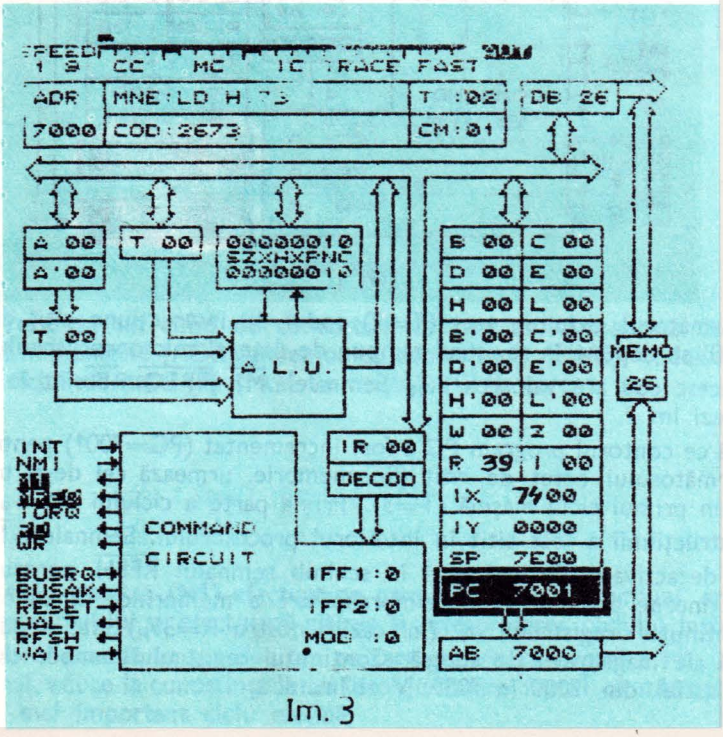
Im.1

În următorul ciclu de tact (T=2) codul de instrucțiune citit de la adresa (PC=7000) se depune în registrul tampon de date al microprocesorului. În cazul de față acest cod are valoarea 26_H. Semnalele $\overline{M1}$, $\overline{MREQ}$, $\overline{RD}$ sînt în continuare active. Vezi Im 2.

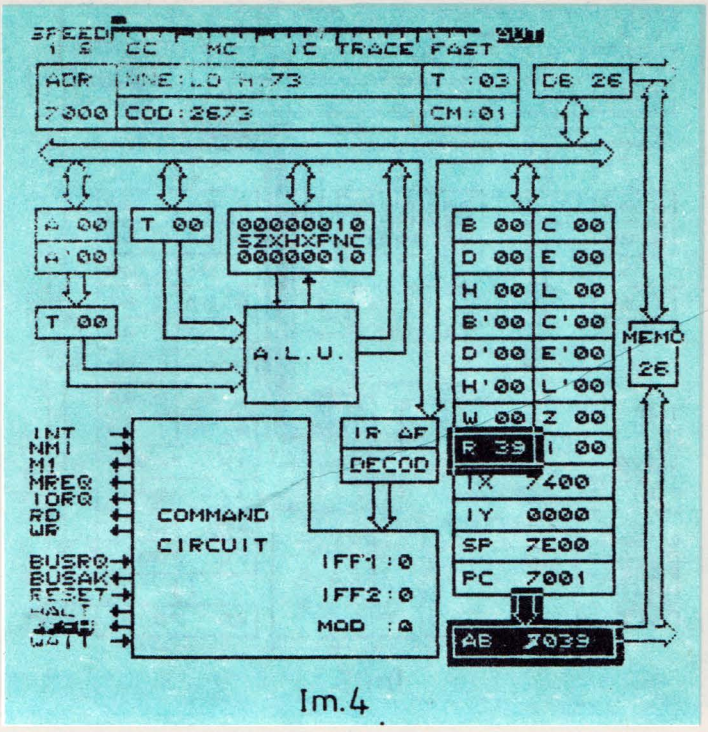
După ce contorul program **PC** a fost incrementat (PC=7001) pentru a indica adresa următorului octet de citit din memorie, urmează cel de-al treilea ciclu de tact din primul ciclu mașină (T=3). Prima parte a ciclului **M1** s-a terminat : codul instrucțiunii a fost citit în interiorul procesorului. Semnalele $\overline{M1}$, $\overline{MREQ}$, $\overline{RD}$ sînt dezactivate. Se activează în schimb semnalul $\overline{RFSH}$, pentru a semnala faptul că începe activitatea de reîmprospătare a memoriilor dinamice. În acest scop conținutul registrului **R** (în cazul nostru $R=39_H$) este depus pe liniile **A0 — A6** ale magistralei de adresă. Conținutul registrului tampon de adresă se schimbă astfel din 7000 în 7039. Vezi Im. 4.


SPEED
1 S CC MC TC TRACE FAST
ADR MNE:LD H,73 T 02
7000 COD:2673 CM:01
DB 26
A 00
A' 00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 00 L 00
B' 00 C' 00
D' 00 E' 00
H' 00 L' 00
W 00 Z 00
R 39 I 00
IX 7400
IY 0000
SP 7E00
PC 7000
AB 7000
MEMO
26
IR 00
DECOD
INT
NMI
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 2


SPEED
1 S CC MC TC TRACE FAST
ADR MNE:LD H,73 T :02
7000 COD:2673 CM:01
DB 26
A 00
A' 00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 00 L 00
B' 00 C' 00
D' 00 E' 00
H' 00 L' 00
W 00 Z 00
R 39 I 00
IX 7400
IY 0000
SP 7E00
PC 7001
AB 7000
MEMO
26
IR 00
DECOD
INT
NMI
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 3



Im.4

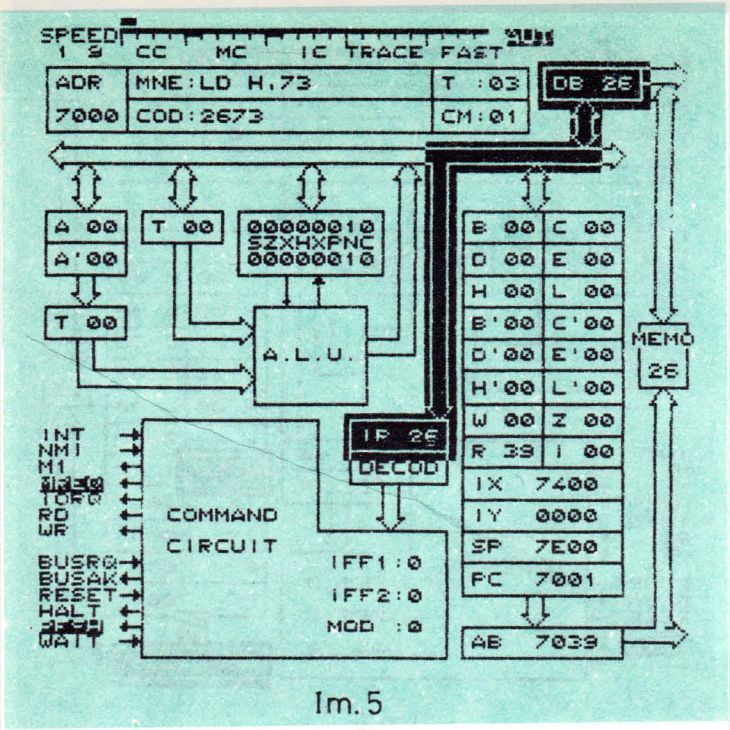
Concomitent cu emiterea adresei de refresh, codul instrucțiunii se depune în registrul de instrucțiune **IR**, pentru a fi interpretat. Este foarte eficientă utilizarea resurselor de timp în acest caz: microprocesorul desfășurînd activități interne, pentru decodificarea codului din **IR**, magistralele externe de date și adrese sînt libere. Ele pot fi folosite în alt scop, de exemplu pentru a se reîmprospăta o eventuală memorie **RAM** dinamică, legată în sistem, ușurînd astfel considerabil munca proiectantului de hardware.

Acum se activează din nou semnalul $\overline{\text{MREQ}}$, care poate fi folosit pentru generarea semnalului $\overline{\text{RAS}}$ (Row Address Strobe), semnal specific al memoriilor dinamice. Menționăm că semnalul $\overline{\text{RFSH}}$ nu putea fi folosit în acest scop, deoarece în momentul activării lui (la începutul lui T=3), adresele de reîmprospătare (A0—A6) nu se puteau considera stabilizate. Astfel constructorii de hardware vor putea folosi semnalul $\overline{\text{RFSH}}$, doar pentru a comuta regimul de lucru al memoriei. Starea curentă a microprocesorului o vedem în Im. 5.

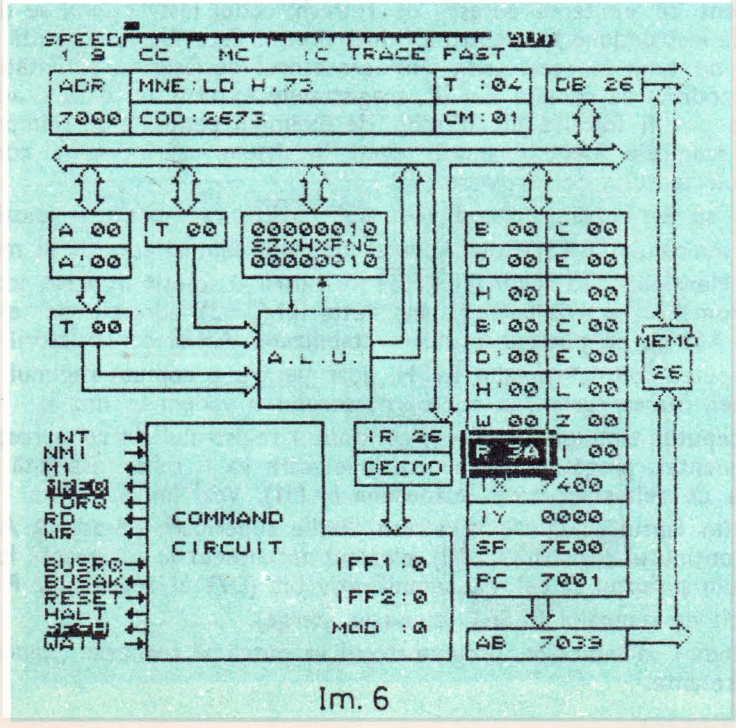
La începutul tactului **T4** (T=4) conținutul registrului de reîmprospătare **R**, este incrementat, pregătindu-se adresa liniei care va fi reîmprospătată în următorul ciclu de reîmprospătare (totdeauna în **M1**). Vezi Im. 6.

Amintim faptul că în ciclul refresh, liniile superioare de adresă **A8 — A15** vor reda conținutul registrului I (!), element necunoscut la elaborarea lui Visible Z80, precum și faptul că cel mai semnificativ bit **(D7)** al registrului **R** nu este afectat decît de semnalul $\overline{\text{RESET}}$ (care-l va șterge).

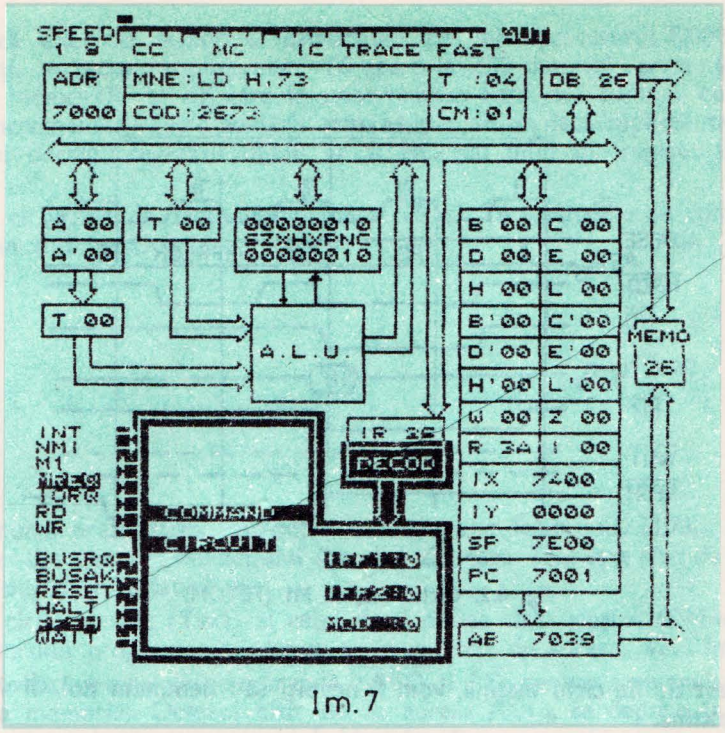
Utilizîndu-l ca indicator, programatorul va putea să realizeze „capcane", cel puțin interesante.


SPEED
1 2 CC MC IC TRACE FAST
ADR
7000
MNE:LD H,73
COD:2673
T :03
CM:01
DB 26
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00 C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 39 I 00
IX 7400
IY 0000
SP 7E00
PC 7001
AB 7039
MEMO
26
IR 26
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 5


SPEED
1 2 CC MC IC TRACE FAST
ADR
7000
MNE:LD H,73
COD:2673
T :04
CM:01
DB 26
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00 C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 3A I 00
IX 7400
IY 0000
SP 7E00
PC 7001
AB 7033
MEMO
26
IR 26
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 6



Im.7

Ciclul mașină **M1** se încheie la sfîrșitul tactului **T4**, cînd codul citit este decodificat. Vezi Im. 7. Acesta este momentul în care microprocesorul a înțeles tot ce are de făcut pentru efectuarea instrucțiunii citite. **(LD H,n)**

Pentru mai multă rigurozitate și pentru a veni în ajutorul celor care doresc să realizeze fizic un sistem cu microprocesorul Z80, redăm în fig. 4.2. diagramele de timp ale semnalelor interesante.

Reținem faptul că ciclul mașină **M1** durează 4 tacți procesor. Sînt numeroase instrucțiuni care pot fi finalizate pe primul front crescător al semnalului de tact, care urmează după **T4**. Aceste instrucțiuni vor fi deci executate într-un singur ciclu mașină. (Din această clasă fac parte toate instrucțiunile de transfer între regiștri **LD r_1, r_2**).

Există însă instrucțiuni care, cu toate că nu mai necesită efectuarea nici unui alt acces la memorie (sau mai general spus nici o altă activitate externă procesorului), nu pot fi executate pe durata ultimului tact **T4** al primului ciclu mașină. Acolo unde necesarul de timp suplimentar este de un ciclu de tact, problema s-a rezolvat amînînd execuția ultimei secvențe pe următorul ciclu mașină, (ciclu care poate face parte din instrucțiunea următoare). Astfel se poate cîștiga durata unui ciclu de tact, care înseamnă un spor de viteză de 20% (4 tacți în loc de 5). Tehnica se numește suprapunerea ciclurilor mașină (sau „furt de ciclu"). Asupra ei vom reveni la tratarea instrucțiunilor logice în *Aventura a 4-a*, acolo unde ea este utilizată frecvent.

Pentru instrucțiunile care necesită 2 tacți suplimentari, ciclul mașină **M1** se prelungește cu 2 tacți. Întrucît în literatura de specialitate nu se amintește

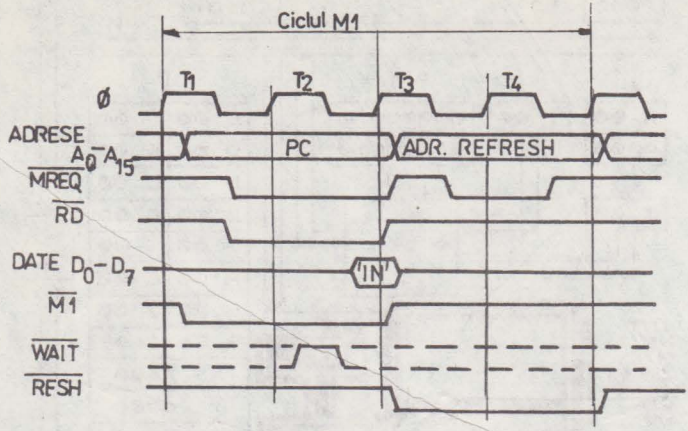


Fig. 4.2. Ciclul mașină M1 (FETCH)

explicit acest tip de ciclu mașină, vom fi nevoiți să-l denumim noi. Îl vom numi ciclu **M1 extins.**

În fig. 4.3. redăm diagrama de timp a semnalelor procesor folosite în acest ciclu.

Instrucțiunile care lucrează cu acest gen de ciclu **M1** sînt cele de incrementare și decrementare a regiștrilor dubli : **INC rr** și **DEC rr**, unde **rr** poate fi **BC, DE, HL, SP, IX, IY.**

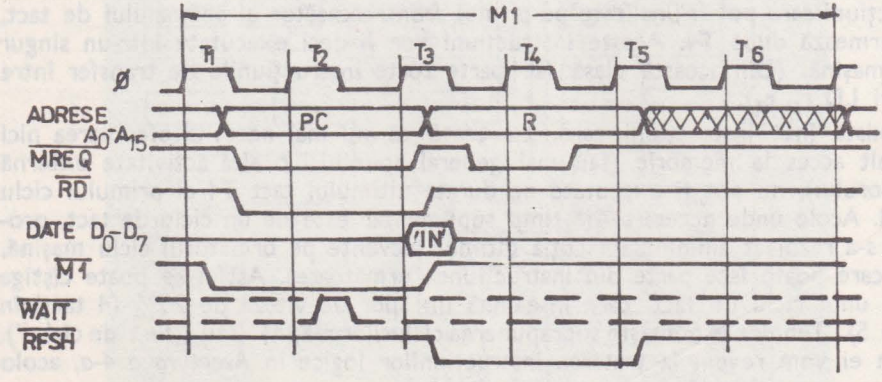


Fig. 4.3. Ciclul M1 extins, avînd 6 cicluri de tact (ex. : INC HL)

Amintim că execuția aceloraşi instrucțiuni pe microprocesorul **I8080** durează doar 5 tacți — Tactul suplimentar **T6** apare la **Z80** datorită prezenței regiștrilor de index **IX** şi **IY** care folosesc resurse hardware comune cu **HL.** (La execuția incrementării/decrementării este nevoie de un tact suplimentar pentru luarea unei decizii : registrul dublu va fi **HL,** sau unul din regiştri index **IX, IY ?**).

Pe durata ciclurilor de tact suplimentare **T5** şi **T6** semnalele de comandă ale procesorului sînt inactive.

4.3. Ciclul maşină de citire din memorie (READ)

Continuăm prezentarea instrucțiunii ilustrate la **fetch : LD H,73$_H$.** Cel de-al doilea octet din cîmpul instrucțiunii (octetul de date 73_H) este citit de la adresa 7001_H şi depus în registrul **H.**

În primul ciclu de tact (T=1) al celui de-al doilea ciclu maşină (CM=2) conținutul contorului program este depus pe magistrala de adrese. Vezi Im. 8. Procesorul activează semnalele de comandă $\overline{\text{MREQ}}$ şi $\overline{\text{RD}}$ pentru a semnala operația de citire a memoriei. Octetul citit de la adresa 7001_H, se depune în registrul tampon de date al microprocesorului. Vezi Im. 9.

Urmează o activitate standard (ce apare în cursul fiecărui ciclu de acces la memorie efectuat cu participarea lui **PC**) incrementarea contorului de adresă, pentru a indica adresa următoarei celule ce va fi accesată. (vezi Im. 10).

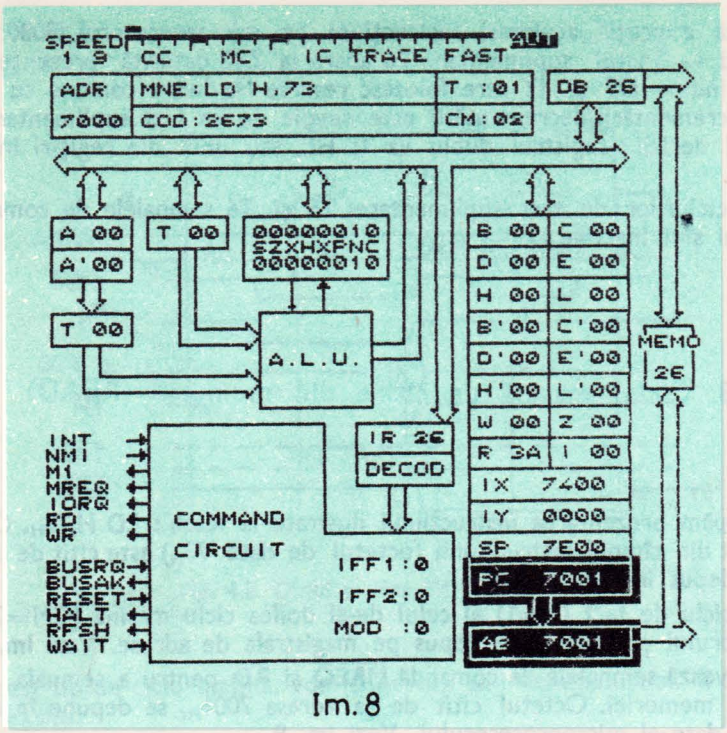
Ciclul **READ** se termină în **T3,** odată cu depunerea octetului citit în registrul intern dorit, (**H** în cazul de față) (vezi Im. 11).

Pentru a corecta micile inexactități de temporizare ale lui **VISIBLE—Z80** redăm celor interesați diagramele de timp, ale ciclului de citire al memoriei **(READ).** (vezi fig. 4.4).

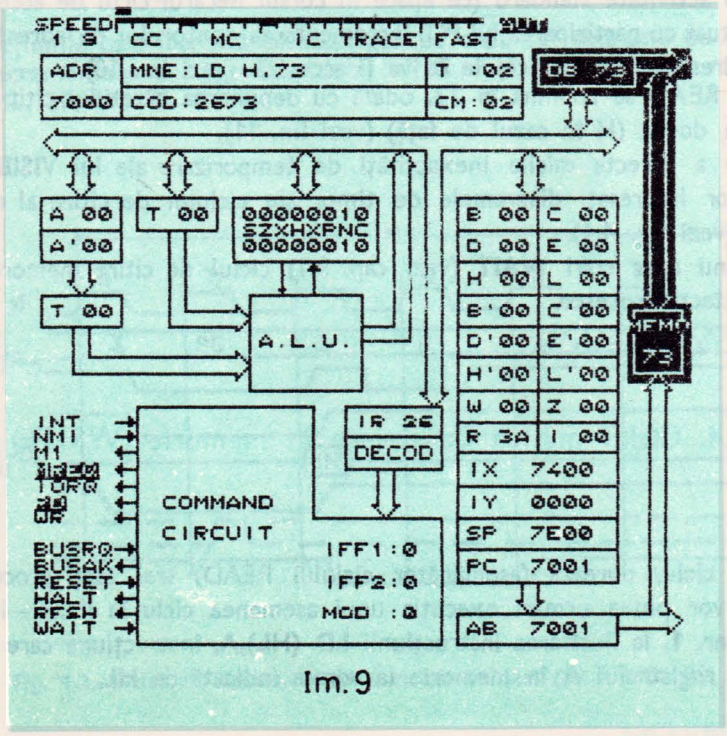
Dacă nu apar stări **WAIT** (vezi cap. 9.1) ciclul de citire memorie READ durează 3 tacți procesor.

4.4. Ciclul maşină de scriere în memorie (WRITE)

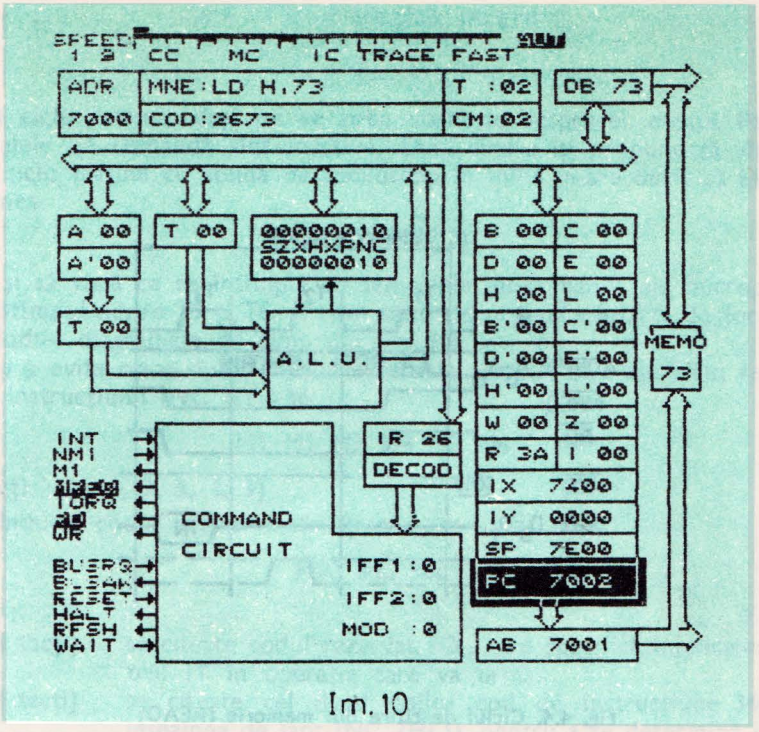
Acest ciclu, durează (asemănător ciclului READ) trei tacți procesor. Cei interesați vor putea urmări execuția unui asemenea ciclu în IM.52—IM.54 din **Aventura nr. 1,** la ilustrarea instrucțiunii **LD (HL),A,** instrucțiune care transferă conținutul registrului **A** în memorie la adresa indicată de **HL.**


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:LD H,73
COD:2673
T :01
CM:02
DB 26
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 3A
I 00
IX 7400
IY 0000
SP 7E00
PC 7001
AB 7001
MEMO
26
IR 26
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

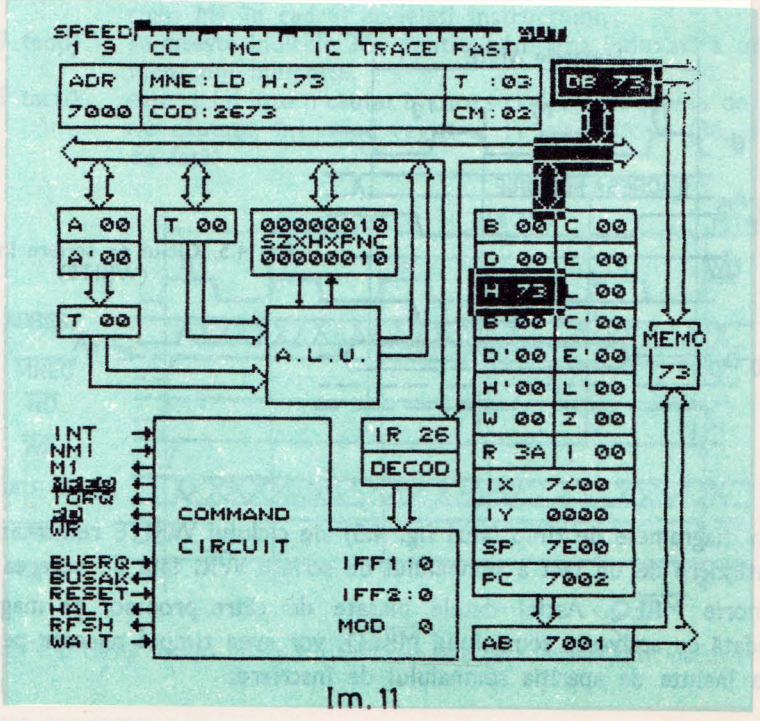
Im.8


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:LD H,73
COD:2673
T :02
CM:02
DB 73
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 3A
I 00
IX 7400
IY 0000
SP 7E00
PC 7001
AB 7001
MEMO
73
IR 26
DECOD
INT
NMI
M1
IORQ
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.9


SPEED
1 2 CC MC IC TRACE FAST
ADR
7000
MNE:LD H,73
COD:2673
T :02
CM:02
DB 73
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 3A I 00
IX 7400
IY 0000
SP 7E00
PC 7002
AB 7001
MEMO
73
IR 26
DECOD
INT
NMI
M1
IORQ
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im. 10


SPEED
1 2 CC MC IC TRACE FAST
ADR
7000
MNE:LD H,73
COD:2673
T :03
CM:02
DB 73
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 73 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 3A I 00
IX 7400
IY 0000
SP 7E00
PC 7002
AB 7001
MEMO
73
IR 26
DECOD
INT
NMI
M1
IORQ
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD 0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im. 11

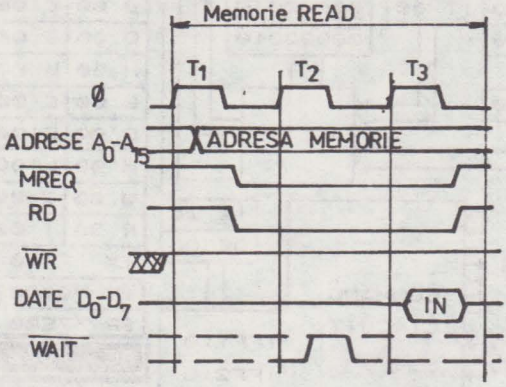


Fig. 4.4. Ciclul de citire din memorie (READ)

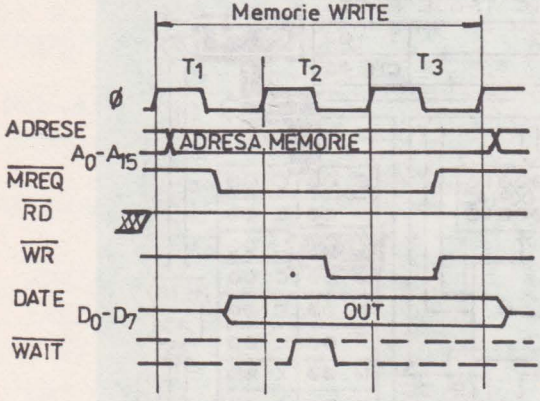


Fig. 4.5. Ciclul de scriere în memorie (WRITE)

Din diagramele de timp (vezi fig. 4.5) ale ciclului **WRITE** remarcăm apariţia cu o întîrziere de un tact a semnalului de scriere $\overline{WR}$, faţă de cererea de acces la memorie $\overline{MREQ}$. Astfel datele plasate de către procesor pe magistrala de date, odată cu activarea semnalului $\overline{MREQ}$, vor avea timpul necesar pentru a se stabiliza înainte de apariţia semnalului de înscriere.

4.5. Ciclul mașină intern

Acest ciclu mașină, deși nu se prea vorbește despre el, există. Pe durata lui, semnalele de comandă sînt inactive. Dacă cineva își propune să vîneze un astfel de ciclu mașină cu sonda de osciloscop în mînă, n-are decît să introducă instrucțiunea

INC (IY + IND)

în buclă, și să vadă ce se întîmplă cu semnalele de comandă ale microprocesorului în răstimpul dintre T1 și T6. Îl vom scuti de această căutare, de loc ușoară, recomandîndu-i diagramele de timp din fig. 4.6.

Pentru a evita orice dubii privind existența acestui ciclu analizăm repartiția de tact a instrucțiunii INC ($IY+36_H$) :

M1 M1 M3 M4 M5 M6

23 tacți (4, 4, 3, 5, 4, 3)

Rememorîndu-se codul acestei instrucțiuni :

FD_H 34_H 36_H

constatăm :

M1 (4 tacți) se citește codul rezervat FD_H care specifică implicarea registrul IY în operația care va urma.

M1 (4 tacți) se citește cel de-al doilea cod de instrucțiune 34_H (care înseamnă de fapt **INC (HL)**), pentru a se determina instrucțiunea de executat. Se observă prezența celui de-al doilea ciclu **M1** în cadrul aceleiași instrucțiuni.

M3 (3 tacți) se citește indicele 36_H pentru adresarea indexată a unui element din memorie.

M4 (5 tacți) este ciclul intern căutat în care se calculează adresa de memorie căutată, însumînd registrul IY cu indicele (36_H în cazul de față).

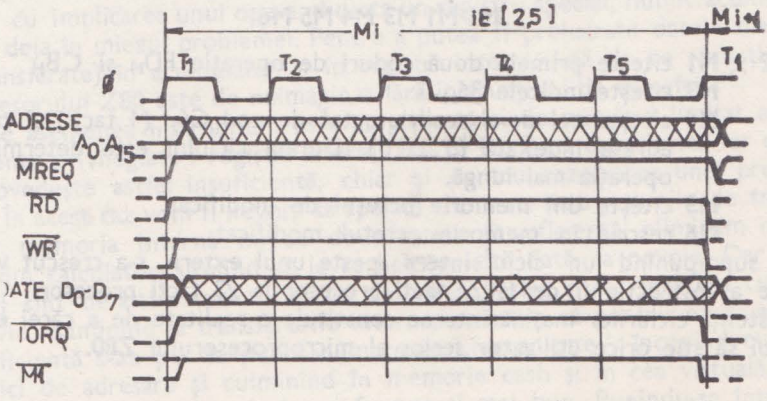


Fig. 4.6. Ciclul mașină intern ; toate semnalele de comandă sînt inactive

M5 (4 tacți) se citește octetul dorit din memorie și se incrementează.
M6 (3 tacți) Se rescrie valoarea incrementată în celula de memorie adresată prin IY + 36_H

Ciclul mașină intern se poate executa și în paralel cu alte activități externe ale procesorului. Astfel se cîștigă timp. Demonstrăm evenimentul cu ajutorul instrucțiunii :

SET 0,(IY+36_H)

cod : **FD_H CB_H 36_H $C6_H$**

Remarcăm prezența în cîmpul instrucțiunii de 4 octeți a 3 coduri de instrucțiuni (FD_H, CB_H, $C6_H$) și a unui octet de date (36_H). De ce oare cel de-al treilea cod ($C6_H$) nu a fost amplasat imediat după CB_H ? Răspunsul este : pentru a cîștiga 4 tacți în execuția instrucțiunii.

După citirea indicelui (36_H) oricum va trebui efectuată adunarea lui la conținutul registrului IY, operație care durează 5 tacți procesor. În paralel se poate citi cel de-al treilea cod de instrucțiune : $C6_H$, care de fapt spune ce anume trebuie făcut, primele două coduri nespunînd decît generalități.

Dacă cel de-al treilea cod de instrucțiune, ($C6_H$) ar fi fost implementat pe poziția a treia distribuția de tacți în instrucțiune ar fi fost :

SET 0,(IY+36_H) caz *ipotetic* : **(4, 4, 4, 3, 5, 4, 3) — 27** tacți

M1 M1 M1 M4 M5 M6 M7

unde : 3 × **M1** ar fi fost afectate pentru citirea celor 3 coduri de instrucțiune
M4 ar fi citit indicele (36_H)
M5 ar fi calculat adresa indexată
M6 ar fi citit din memorie octetul de modificat
M7 ar fi reînscris în memorie octetul modificat

În realitate avem :

SET 0,(IY+36_H) caz *real* : **(4, 4, 3, 5, 4, 3) — 23** tacți

M1 M1 M3 M4 M5 M6

unde : 2 × **M1** citește primele două coduri de operație FD_H și CB_H
M3 citește indicele 36_H
M4 citește cel de-al treilea octet de cod $C6_H$ (4 tacți) și calculează adresa indexată (5 tacți). Durata ciclului este determinată de operația mai lungă.
M5 citește din memorie octetul de modificat.
M6 rescrie în memorie octetul modificat.

Astfel, suprapunînd un ciclu intern peste unul extern, s-a crescut viteza de execuție a instrucțiunii de la 27 tacți procesor la 23 tacți procesor.

Existența ciclurilor mașină interne constituie o realitate de a cărei existență va trebui să știe orice utilizator serios al microprocesorului **Z80.**

5
INSTRUCȚIUNI DE TRANSFER

Sînt oare transferurile un scop în sine ? Cunoscînd faptul că din cele 696 de instrucțiuni ale microprocesorului **Z80,** 154 nu fac altceva decît mută octeții în interiorul procesorului și între procesor și memorie, am fi tentați să credem că da. Acceptînd transferurile ca o expresie a mișcării, gîndurile ne îndeamnă la asocieri mai ample. Mișcarea este o formă de existență. O formă de existență „a tot ceea ce este viu", însăși a materiei. Dar cum, spre deosebire de materie, microprocesorul este un produs al omului, să nu ne grăbim cu răspunsul; nici dacă una din cele două funcții majore ale calculatoarelor electronice, care s-au cristalizat în cele 4 decenii de existență ale lor, vehicularea de date, se constituie în proporție de 90% din transferuri. Oricît de sofisticate ar fi calculatoarele din zilele noastre, datorită însăși genezei lor, ele nu pot nicidecum întreprinde acțiuni cu un „raison" mai profund decît cele pe care le permite imaginația oricum limitată a creatorului lor, omul. Așa stînd lucrurile, să încercăm să depistăm raționamentul care impune includerea în setul de instrucțiuni al oricărei unități centrale existente a unui subset bogat de instrucțiuni de transfer. Căutările noastre nu trebuie să fie îndelungate.

Reamintindu-se doar faptul că operațiile aritmetice și logice se desfășoară totdeauna cu implicarea unui operand dintr-un registru special, numit acumulator, ne situăm deja în miezul problemei. Pentru a putea fi prelucrate datele trebuiesc aduse (transferate) în acumulator și în regiștri interni. Iată de ce funcționarea microprocesorului **Z80** este de neimaginat fără instrucțiuni de transfer.
Cererea de asemenea instrucțiuni este amplificată și de numărul limitat al unor alte elemente privilegiate : regiștri interni. Foarte frecvent capacitatea lor de stocare se dovedește astfel insuficientă, chiar și pentru rezolvarea unei probleme mărunte. În acest caz vom fi nevoiți să apelăm din nou instrucțiunile de transfer. Dacă nici memoria internă nu se dovedește a fi suficientă, eliminăm o parte din date din calculator, urmînd să le rechemăm altă dată, la nevoie. Dar aceste transferuri sînt de altă natură : ele sînt de intrare/ieșire și fac legătura cu lumea externă, vie. Rămînînd la transferurile interne, amintim că pentru efectuarea lor cît mai eficientă s-au pus la punct numeroase „mecanisme", începînd cu diversele tehnici de adresare și culminînd în memorie cash și în cea virtuală. Adică tot ce a realizat tehnica de calcul mai frumos și mai bun. Punîndu-se întrebarea dacă instrucțiunile de transfer ar putea fi totuși eliminate, răspunsul îl obținem reconsiderînd cauzele care au dus la apariția lor : inegalitatea elementelor com-

ponente. Dacă s-ar putea realiza o structură în care toți membri să fie egali, dotați cu aceleași resurse, ar fi posibil. Dar așa cum o demonstrează istoria, ar fi cel puțin păcat. Un model astfel realizat ar rămîne unul utopic. Atît.

Acceptînd ideea că instrucțiunile de transfer reprezintă un rău necesar, să vedem ce ne oferă ele.

În funcție de cantitatea informațională pe care o vehiculează distingem 3 clase de instrucțiuni de transfer :

— **instrucțiuni de transfer de 8 bit** — vehiculează 1 octet
— **instrucțiuni de transfer de 16 bit** — vehiculează 2 octeți
— **instrucțiuni de transfer de blocuri de date** — vehiculează 1 ÷ 65536 octeți.

Ca acțiune generală notăm că în majoritatea lor copleșitoare, instrucțiunile de transfer acționează *unidirecțional* : informația se preia dintr-un loc denumit sursă și se depune în alt loc numit destinație. Combinațiile **sursă-destinație** pot fi :

— **registru intern — registru intern,**
— **registru intern — memorie**
— **memorie — registru intern**
— **memorie — memorie**

Acest din urmă tip de transfer apare la instrucțiunile care vehiculează blocuri de date, transferîndu-le dintr-o zonă de memorie în alta.

Cu puține excepții mnemonica instrucțiunilor de transfer este

LD (load-încarcă)

Forma generală a instrucțiunii scrisă în limbaj de asamblare este

LD destinație, sursă

Există însă și instrucțiuni care declanșează transferuri *bidirecționale*. În acest caz este impropriu a vorbi de sursă și destinație, întrucît ambele elemente implicate în transfer sînt atît surse cît și destinații, conținutul lor interschimbîndu-se.

Aceste instrucțiuni au mnemonica generică EX, forma lor, scrisă în limbaj de asamblare fiind :

EX element1, element2

Toate instrucțiunile de acest gen operează pe grupe de 2 octeți (16 bit), totalul lor cifrîndu-se la 6.

Tot în clasa instrucțiunilor de transfer pe 16 bit găsim și cele notate cu **PUSH** respectiv **POP**. Prin natura transferului efectuat, ele vor putea fi considerate ca etalon de load (transfer) indirect între regiștri dubli și memorie. Prin alocarea acestor mnemonici, diferite de **LD**, elaboratorii microprocesorului evidențiază folosirea unui registru de adresă special (indicatorul de stivă **SP**) și o o tehnică (detaliu) specială.

Numărul total al acestor instrucțiuni este de 12.

Instrucțiunile de transfer a blocurilor de date, ultimele a căror mnemonici diferă de **LD** au fost mai fidele : ele păstrează în numele lor — **LDD, LDI, LDDR, LDIR** — cele două litere care le trădează apartenența.

În rest instrucțiunile de transfer sînt destul de pașnice. Viața lor este o carte deschisă (totul se înțelege din enunțul instrucțiunii) ele neafectînd nici un indicator de condiție (flag).

Și totuși pentru ca regula să fie confirmată există și aici 4 excepții, instrucțiunile **LD A,I ; LD A,R ; EX AF, AF' ; POP AF.**

5.1. Aventura 1 : transferuri de 8 bit

Să recurgem acum la **VISIBLE—Z80** pentru a întreprinde o incursiune în lumea instrucțiunilor care transferă cîte un octet. O aventură pe parcursul căreia vom încerca să dezlegăm toate tainele, oricît de ascunse, ale acestei clase de instrucțiuni.

```
7000  LD  H,73          ;Im 1—Im11   (CC)
7002  LD  A,71          ;Im12—Im13   (MC)
7004  LD  B,A           ;Im14        (MC)
7005  LD  I,A           ;Im15—Im16   (MC)
7007  LD  A,(BC)        ;Im17—Im18   (MC)
7008  LD  (7406),A      ;Im19—Im22   (MC)
700B  LD  D,(IX+06)     ;Im23—Im49   (CC)
700E  LD  (HL),75       ;Im50—Im54   (MC şi CC)
```

Şirul de instrucțiuni de mai sus este doar o secvență. Ea nu poate fi numită program, datorită faptului că nu are funcționalitate în sine. (Dacă nu cumva atașăm această noțiune fenomenului că spre deosebire de momentul lansării cînd doar contorul program (PC=7400) diferă de zero, la sfîrșit conținutul regiștrilor dubli se aliniază și el : HL=7300 ; DE=7200 ; BC=7100. Dar asta este artă pentru artă.).

În coloana din stînga avem adresele de început, în coloana a 2-a enunțul instrucțiunilor în limbaj de asamblare, iar în cea de a 3-a, care reprezintă comentarii, numărul imaginilor generate de **VISIBLE—Z80** pe care prezentăm instrucțiunea cu pricina. Specificația din paranteză **CC**, sau **MC** semnalează modul de lucru al simulatorului **VISIBLE—Z80.**

CC — **clock cycle** — regimul cel mai detailat, tact cu tact.

MC — **machine cycle** — regimul în care se face o singură vizualizare la sfîrșitul fiecărui ciclu mașină.

Toate numerele sînt hexazecimale.

Instrucțiunea **LD H,73** care este o instrucțiune de încărcare imediată a unui registru intern, am prezentat-o (Im.1—Im.11) în cap. 4. Rememorîndu-se ultima imagine Im.11 observăm că data (73_H) este „aruncată" peste registrele **B,C,D,E** în **H**. Fenomenul este înșelător, căci în realitate magistrala internă de date ajunge la fiecare registru. Din motive de lipsă de spațiu, în **VISIBLE—Z80** nu a fost reprezentat racordul fiecărui registru în parte la magistrala internă de date, ci doar pentru întregul bloc de regiștri.

Remarca 1 : Dacă în cursul execuției unei instrucțiuni, fie ea de transfer sau nu, oricare din regiștri de uz general este implicat, selectarea lui se va face printr-un multiplexor (de asemenea nereprezentat de **VISIBLE—Z80**), pe baza unei adrese codificate pe 3 bit, preluată din însuși codul instrucțiunii. Adresele regiștrilor interni sînt :

r_2	r_1	r_0	reg.
0	0	0	B
0	0	1	C
0	1	0	D
0	1	1	E
1	0	0	H
1	0	1	L
1	1	0	
1	1	1	A

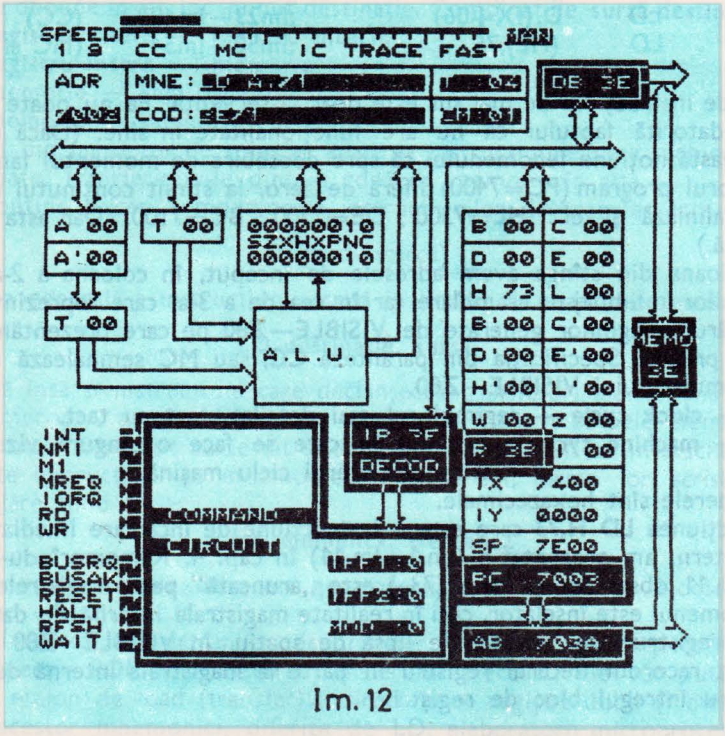
În cazul instrucțiunilor de încărcare imediată a regiștrilor simpli formatul octetului de cod este :

$$0\ 0\ r_2\ r_1\ r_0\ 1\ 1\ 0$$

Biții $b_7(=0)$, $b_6(=0)$, $b_2(=1)$, $b_1(=1)$, $b_0(=0)$ definesc grupa de instrucțiuni, iar $b_5\ (=r_2)$, $b_4(=r_1)$ și $b_3(=r_0)$ specifică registrul dorit.

Pentru registrul H (100) rezultă codul **00100110** (26_H), adică cel pe care-l cunoaștem din imaginile prezentate. Deducem cu ușurință codul următoarei instrucțiuni din listă : **LD A,71**, care face parte din aceeași grupă cu cea precedentă. Registrul A are adresa : **111**. Codul instrucțiunii de încărcare imediată va fi deci **00111110** adică $3E_H$.

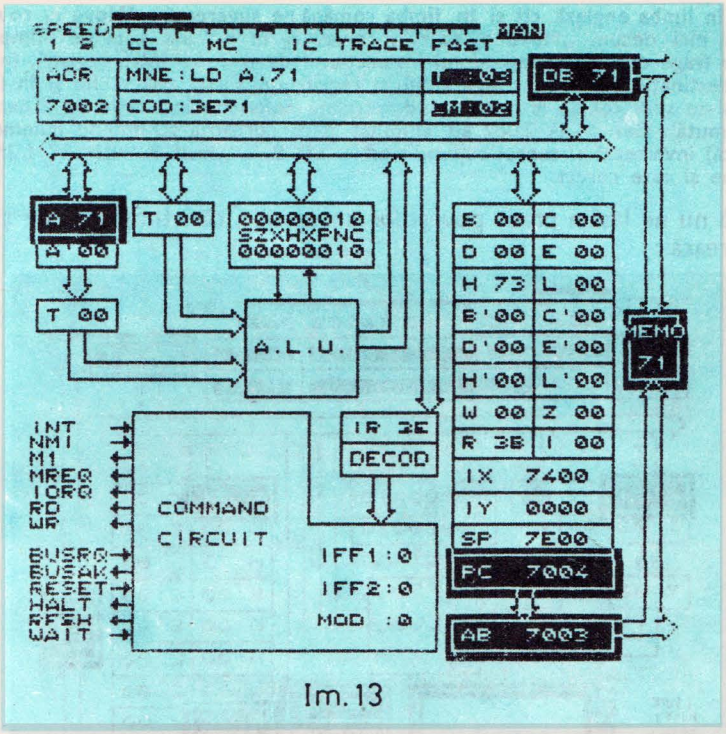
Execuția instrucțiunii fiind similară cu cea precedentă, folosim prilejul de a-l familiariza pe cititor cu imaginile generate de **VISIBLE—Z80** în modul **MC** (ciclu mașină).



Im. 12

Indicatorul de viteză (și mod de lucru) al simulatorului este poziționat pe **MC**. La sfîrșitul primului ciclu mașină, cel de citire al codului de instrucțiune din memorie (CM=1 ; T=4) se indică, vizualizîndu-le în video invers, toate elementele afectate de acel ciclu.

Remarca 2 : în ciclul mașină **M1** (FETCH) elementele afectate par a fi mereu aceleași, indiferent de instrucțiunea executată : contorul program (PC), bufferul de adrese (AB), magistrala de adrese (A0—A15), memoria, magistrala de date (D0—D7), bufferul de date (DB), registrul instrucțiune (IR), registrul refresh (R) și circuitul de comandă, în frunte cu decodificatorul său. Elementele menționate se activează într-adevăr de fiecare dată. Instrucțiunile care se execută într-un singur ciclu mașină, au și elemente suplimentare, așa cum vom vedea pe parcurs.



Im.13

Instrucțiunea **LD A,71** se termină cu cel de-al doilea ciclu mașină, un ciclu READ (CM=2 ; T=3). În Im.13 se vede cum octetul citit de la adresa 7003 (71_H) se depune în registrul **A**.

Ați remarcat desigur faptul că în modul de lucru ciclu mașină **(MC)** semnalele de comandă nu se mai activează. Am renunțat la ele pentru a nu îngreuna inteligibilitatea imaginii. Între timp contorul program indică deja adresa instrucțiunii următoare :

7004 LD B,A ;Im. 14 (MC)

Instrucțiunea de față este o instrucțiune de transfer între regiștri. Ea se codifică pe un singur octet : 47_H.

Spre deosebire de cele două instrucțiuni prezentate mai sus, în cazul transferurilor între regiștri, atît elementul destinație cît și cel sursă se specifică implicit, prin codul binar al instrucțiunii. Structura lor este :

$$0\ 1\ d_2\ d_1\ d_0\ s_2\ s_1\ s_0$$

Primii doi bit — $b_7(=0)$, $b_6(=1)$ semnalează faptul că avem de-a face cu o instrucțiune de transfer între regiștri. Pe biții mai puțin semnificativi $b_5 - b_0$ apare adresa registrului destinație $b_5 - b_3$ și apoi cea a registrului sursă $b_2 - b_0$

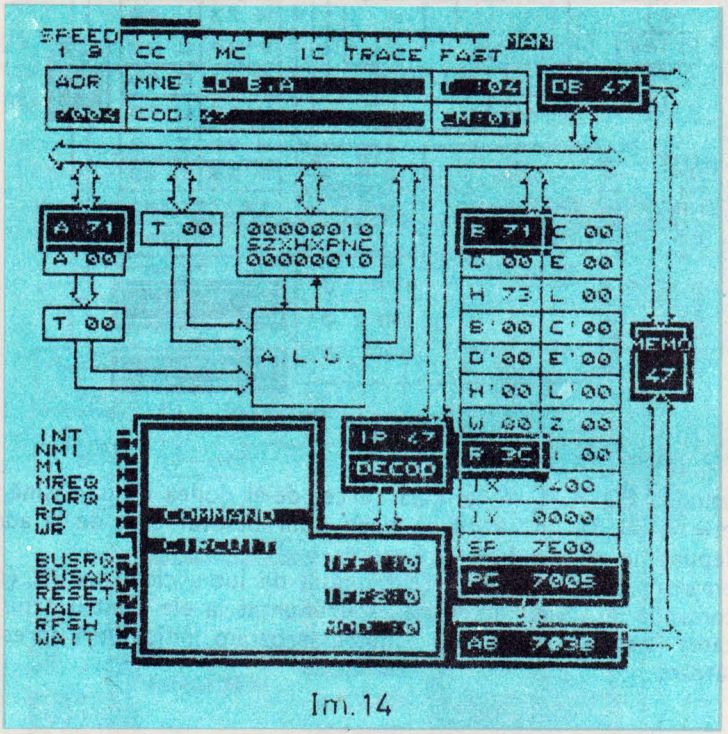
Remarca 3 : Mulți au întrebat de ce a ales firma Intel forma generică a instrucțiunii sale ca fiind :

MOV r_1, r_2

De ce nu se specifică prima dată sursa și apoi destinația ?

Atît citirea în limba engleză cît şi în limba română ne sugerează : "Move r_1 to r_2", „mută r_1 în r_2" şi nici decum „Move r_2 to r_1", „mută r_2 în r_1" aşa cum se întîmplă de fapt. „Păcatul" se trage din structura codului instrucţiunii de transfer între regiştri, în care codul registrului destinaţie apare în stînga celui al registrului sursă. Aşa să fie şi în limbajul de asamblare, şi-au spus cei de la Intel, avînd totodată nefericita inspiraţie de a denumi acţiunea move (mută). Cei de la Zilog au eliminat neconcordanţa schimbînd mnemonica : LD. Load (încarcă) inversează şi sensul acţiunii : astfel LD B,A, „load B with A" („încarcă B cu A) sună bine şi este corect.

Dar să nu ne lăsăm pradă poveştilor cinegetice, căci timpul trece şi **VISIBLE —Z80** lucrează :



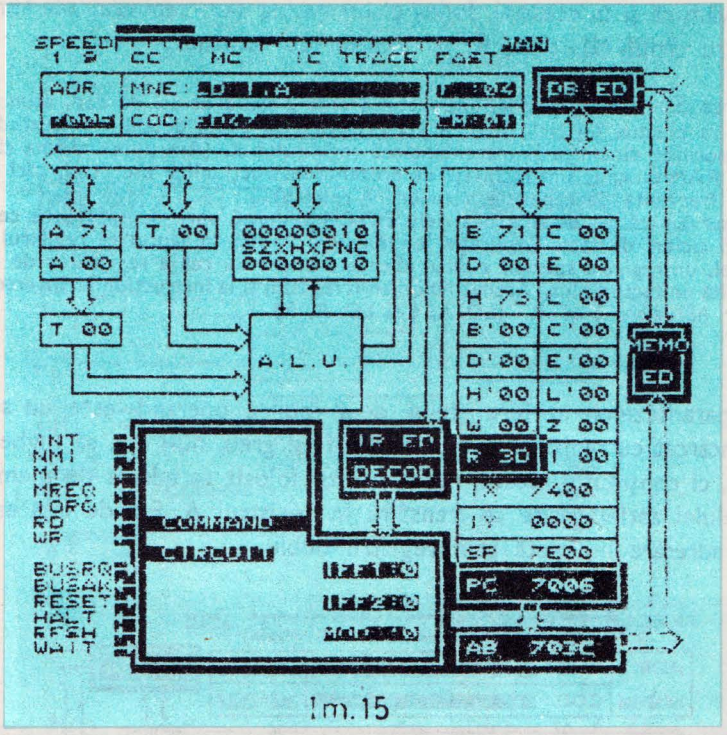
Im. 14

LD r_1, r_2 fiind instrucţiuni cu un singur ciclu maşină în starea CM=1, T=4 LD B,A s-a şi terminat, şi aşa cum vedem în Im.14 registrul **B** conţine deja aceeaşi valoare ca şi **A**, nu 00 cum fusese înainte.

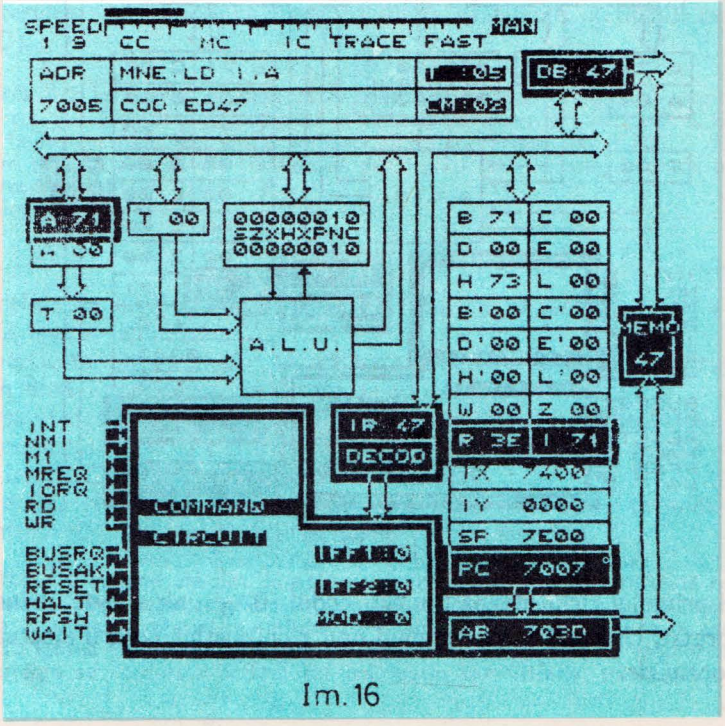
Remarca 4 : La toate instrucţiunile de transfer unidirecţional, conţinutul elementului sursă (registru sau celulă de memorie) rămîne neschimbat. Conţinutul iniţial al elementului destinaţie (registru sau celulă de memorie) se pierde, peste el înscriindu-se conţinutul sursei

```
7005    LD    I,A          ;Im.15—Im.16    (MC)
```

Deşi un transfer între doi regiştri interni ai microprocesorului, — această instrucţiune permite înscrierea vectorului de întrerupere în registrul dedicat **I** —, necesarul ei de timp şi memorie este de cel puţin două ori mai mare decît cel al transferurilor între regiştri de uz general.
In momentul (CM=1, T=4) se termină primul ciclu maşină, odată cu decodificarea codului ED_H, care este un cod rezervat. Urmează citirea codului, propriu-zis : 47_H.


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE: LD A,A
COD: 7F
DB ED
A 71
A' 00
T 00
00000010
SZXHXPNC
00000010
B 71
C 00
D 00
E 00
H 73
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 00
Z 00
R 3D
I 00
IX 7400
IY 0000
SP 7E00
PC 7006
AB 703C
MEMO
ED
A.L.U.
IR ED
DECOD
COMMAND
CIRCUIT
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.15


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
7005
MNE: LD I,A
COD: ED47
DB 47
A 71
T 00
00000010
SZXHXPNC
00000010
B 71
C 00
D 00
E 00
H 73
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 00
Z 00
R 3E
I 71
IX 7400
IY 0000
SP 7E00
PC 7007
AB 703D
MEMO
47
A.L.U.
IR 47
DECOD
COMMAND
CIRCUIT
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

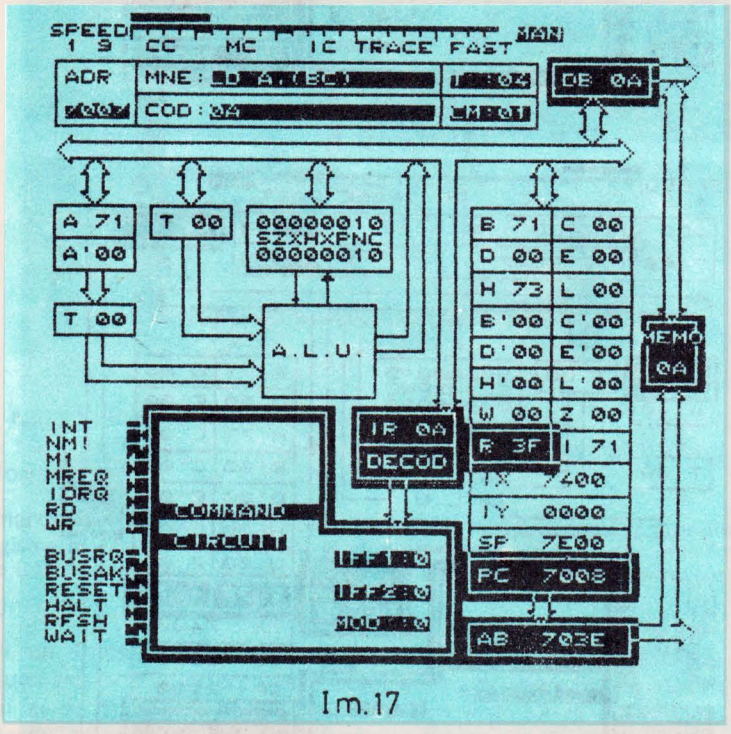
Im.16

Observăm că și în acest al doilea ciclu mașină s-au semnalat elementele specifice pentru ciclul **M1 : IR, R, și circuitul de comandă.**

Remarca 5 : Dacă o instrucțiune încorporează 2 sau 3 octeți de cod, atunci la citirea fiecăruia se va efectua un ciclu mașină **M1** (FETCH) cu toate detaliile, incluzînd reîmprospătarea memoriilor dinamice și incrementarea registrului **R.** (Am putea afirma că circuitele de memorie **RAM** dinamic sînt încîntate de prezența instrucțiunilor multicod, căci executarea lor mărește frecvența de repetiție a ciclurilor refresh).
In Im.16 observăm că cel de-al doilea ciclu FETCH a durat, spre deosebire de cele întîlnite pînă acum 5 cicluri de tact. Explicația fenomenului constă în faptul că registrul **I** fiind, ca și **R** de altfel, un registru special, a cărui abordare diferă de cea a regiștrilor de uz general, acesta necesită un tact în plus. Același fenomen îl regăsim și la instrucțiunile inverse, de citire, a *regiștrilor hardware* **I** și **R.** (**LD A, I** și **LD A, R**).

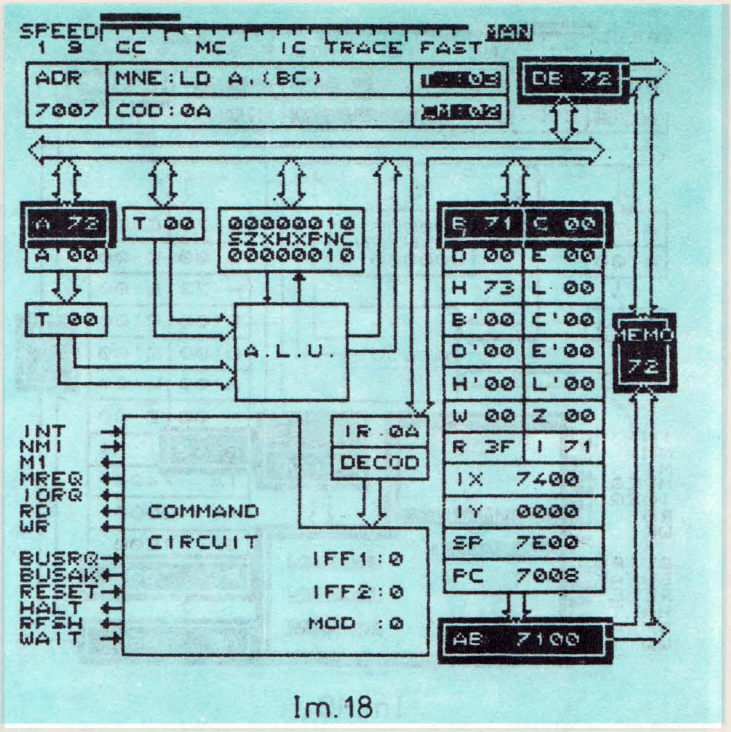
```
7007        LD A, (BC)              ;Im.17—Im.18    (MC)
```

Prezența parantezelor în jurul celui de-al doilea operand este un semn că **A** nu va fi încărcat cu registrul dublu **BC** (ar fi și greu, nu-i așa, să înghesui 16 bit în 8 bit ?), ci conținutul acestui registru este folosit ca adresă de memorie. Conținutul celulei astfel găsite se transferă în registrul **A.** Ex. de față este un caz clasic de adresare indirectă prin registru dublu.



Im.17

La sfîrșitul primului ciclu mașină (Im.17), codul $0A_H$, unicul byte al instrucțiunii, este interpretat. Urmează inserarea unui nou ciclu mașină pentru efectuarea transferului propriu-zis.



Im.18

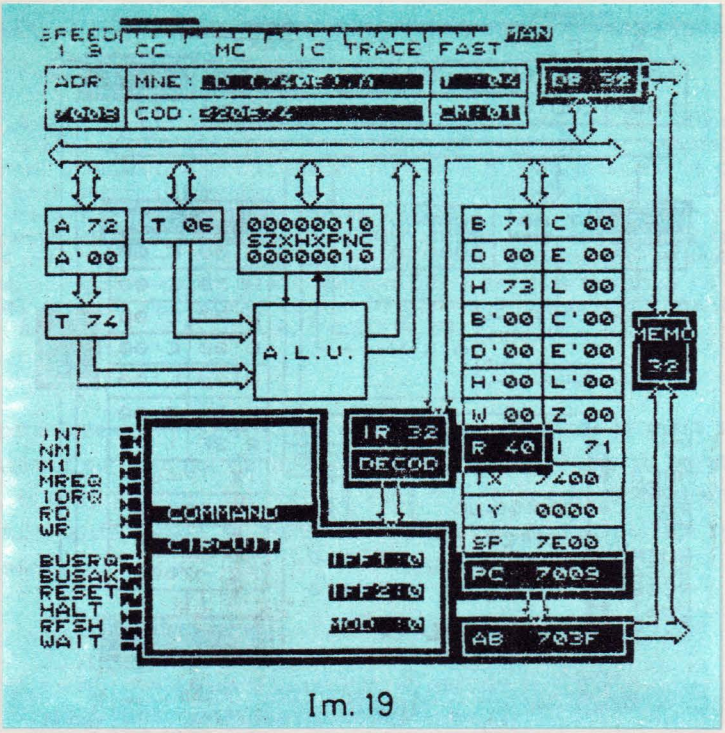
De la adresa 7100_H s-a citit valoarea 72_H care s-a depus în **A.** (Aici începe pelerinajul octetului 72_H care, după lungi peripeţii, va ajunge în finalul aventurii în registrul D).

Remarca 6 : Încărcarea şi salvarea indirectă a acumulatorului este posibilă folosind oricare din cei trei regiştri dubli **BC, DE,** şi **HL.** Ceilalţi regiştri **(B, C, D, E, H, L)** nu pot fi implicaţi în instrucţiuni de transfer indirect, folosind regiştri **BC** şi **DE,** ci doar registrul **HL.** Reţinem deci că atît **A,** cît şi **HL** sînt privilegiaţi faţă de colegii lor de breaslă **B, C, D, E, H, L** respectiv **BC** şi **DE.**
(ex. : instrucţiunea **LD B, (DE)** nu există).

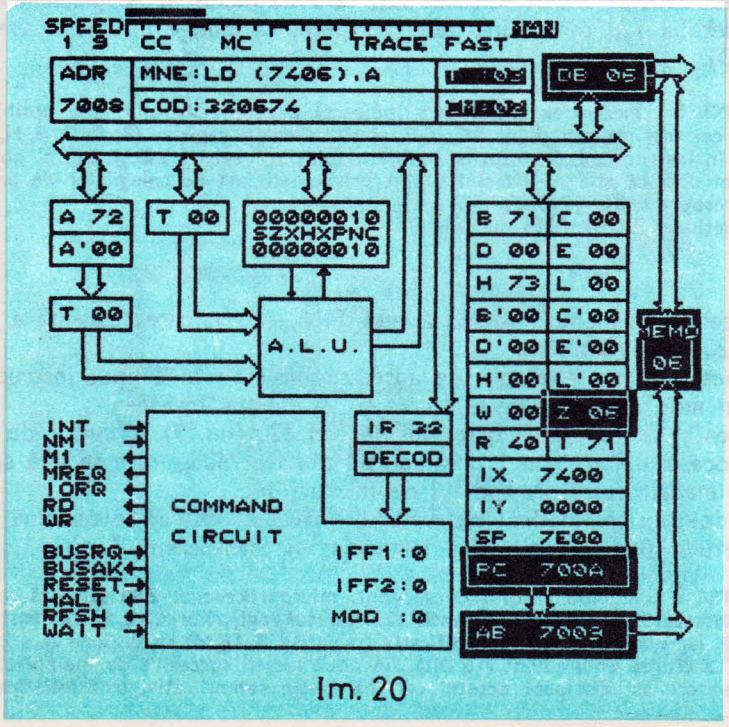
7008 LD (7406),A ; Im.19—Im.22 (MC)

Încărcăm celula de memorie avînd adresa 7406_H, cu valoarea curentă a acumulatorului (72_H — cel amintit).
Instrucţiunea este un exemplu de adresare directă : în cîmpul instrucţiunii se specifică în mod explicit adresa la care se va efectua transferul.
După citirea şi decodificarea octetului de cod 32_H (Im.19), circuitul de comandă al microprocesorului trece la citirea adresei căutate 7406_H, care se află specificată în octeţii imediat următori codului de instrucţiune.
Cel de-al doilea ciclu maşină (ciclu READ în cazul nostru) se termină cu încărcarea octetului 06_H în registrul intermediar şi transparent **Z.**

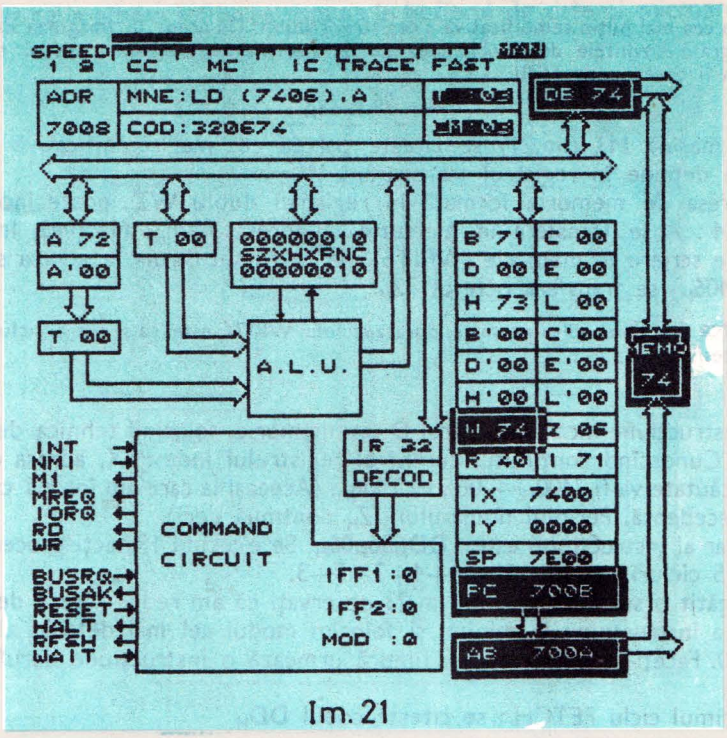
Remarca 7 : În mod absolut consecvent, microprocesorul **Z80** salvează regiştri dubli astfel încît **octetul mai puţin semnificativ** (conţinutul regiştrilor **C, E, L**) se înscrie la adresa nominalizată, iar **octetul cel mai semnificativ** (registrul **B, D, H**) se depune la adresa imediat superioară. La citirea unui cuvînt (16 bit) din două locaţii succesive de memorie, conţinutul adresei inferioare se consideră octetul cel mai puţin semnificativ, transferîndu-se ca atare


SPEED
1 S
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE :
COD :
DE 32
A 72
A'00
T 06
00000010
SZXHXPNC
00000010
T 74
A.L.U.
B 71
C 00
D 00
E 00
H 73
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 40
I 71
IX 7400
IY 0000
SP 7E00
PC 7009
AB 703F
MEMO
32
IR 32
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT

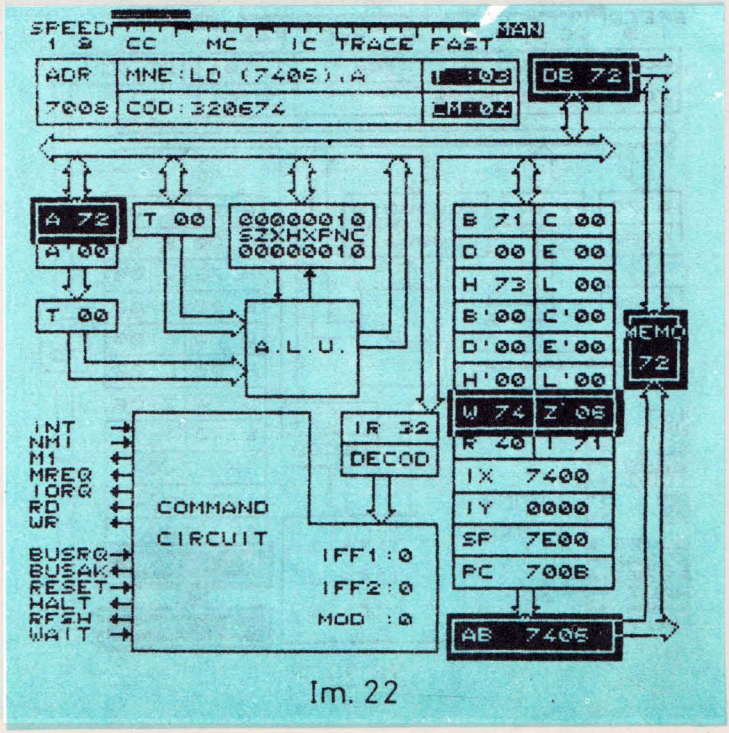
Im. 19


SPEED
1 S
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE:LD (7406),A
COD:320674
DE 05
A 72
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 71
C 00
D 00
E 00
H 73
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 06
R 40
I 71
IX 7400
IY 0000
SP 7E00
PC 700A
AB 7003
MEMO
06
IR 32
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 20


SPEED
1 2
CC
MC
IC
TRACE
FAST
MAN
ADR
7008
MNE:LD (7406),A
COD:320674
DB 74
A 72
A'00
T 00
00000010
SZXHXPNC
00000010
B 71
C 00
D 00
E 00
H 73
L 00
B'00
C'00
D'00
E'00
H'00
L'00
MEMO
74
W 74
Z 06
R 40
I 71
IX 7400
IY 0000
SP 7E00
PC 700B
AB 700A
T 00
A.L.U.
IR 32
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 21


SPEED
1 2
CC
MC
IC
TRACE
FAST
MAN
ADR
7008
MNE:LD (7406),A
COD:320674
DB 72
A 72
A'00
T 00
00000010
SZXHXPNC
00000010
B 71
C 00
D 00
E 00
H 73
L 00
B'00
C'00
D'00
E'00
H'00
L'00
MEMO
72
W 74
Z 06
R 40
I 71
IX 7400
IY 0000
SP 7E00
PC 700B
AB 7406
T 00
A.L.U.
IR 32
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 22

în jumătatea cea mai puțin semnificativă a regiștrilor dubli. De aceea în imaginea de memorie a unui program cuvintele de 16 bit apar cu octeți inversați. În cazul nostru mnemonicii **LD (7406),A** îi corespunde codul

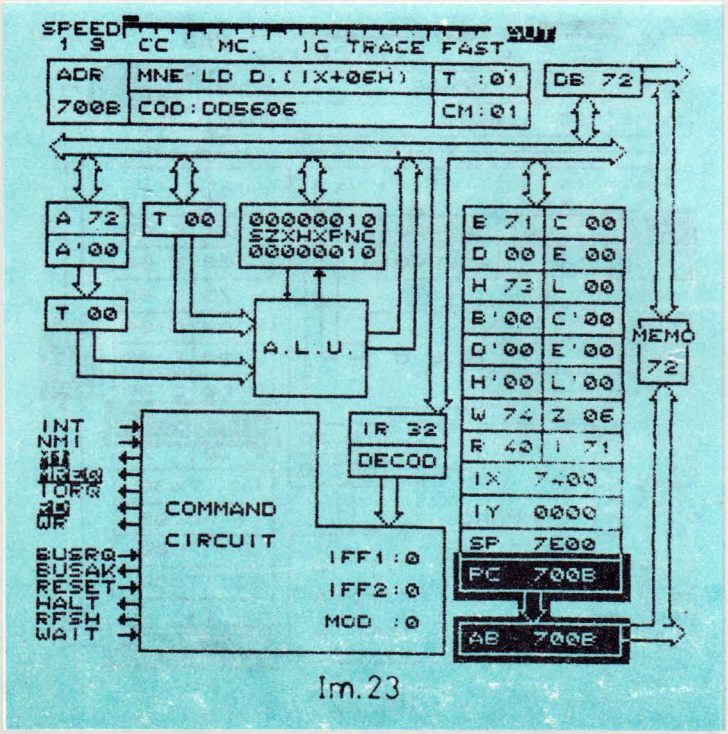
$$32_H\ 06_H\ 74_H$$

În ciclul mașină **M3** (Im.21) se citește partea cea mai semnificativă a adresei, 74_H, și se depune în registrul transparent **W**.
Avînd adresa de memorie formată în registrul dublu **WZ**, poate începe ciclul mașină **M4 : A** se transferă în memorie la adresa 7406_H. **M4** (vezi Im.22) este un ciclu de scriere în memorie (WRITE), pe parcursul căruia la adresa specificată de **AB** (7406_H) se transferă octetul 72_H.

Remarca 8 : În condiții normale, durata ciclului WRITE este, ca și cea a ciclului READ, 3 tacți procesor.

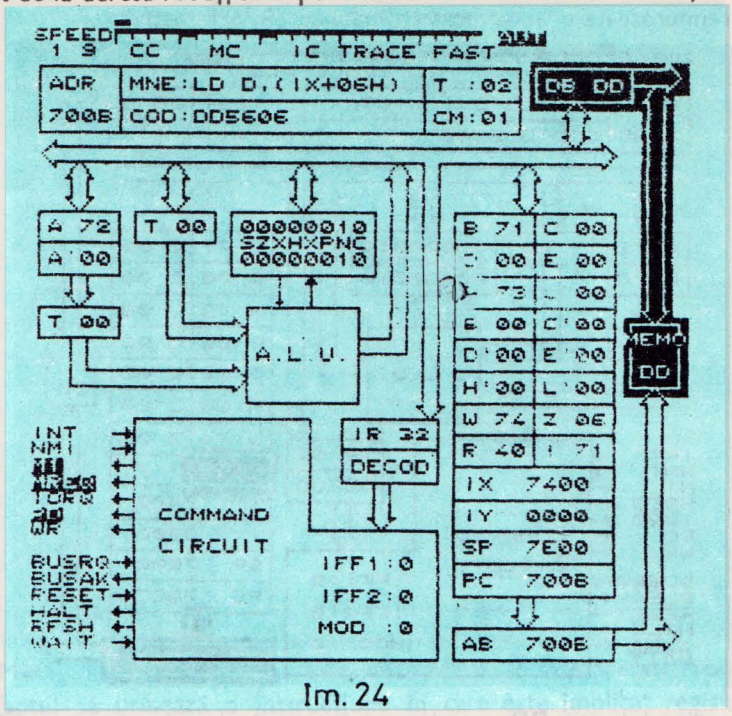
700B LD D,(IX+06H) ; Im.23—Im.49 (CC)

Această instrucțiune încarcă registrul **D** din memorie, folosind tehnica de adresare indexată. Cunoscînd conținutul curent al registrului index **IX**, adresa celulei de memorie căutate va fi $7400_H + 06_H = 7406_H$. (Aceeași la care am înscris cu instrucțiunea precedentă. Periplul numărului 72_H continuă deci).
Codul binar al instrucțiunii este : $DD_H56_H06_H$. Se execută 19 tacți procesor repartizați pe 5 cicluri mașină $19 = 4+4+3+5+3$.
V-am pregătit o surpriză. Privind Im.23 observați că am redus viteza de simulare **(SPEED)** a instrucțiunii la minim, și folosim modul cel mai detailat al lui **VISIBLE—Z80.** Faceți-vă deci comozi, fiindcă urmează o instrucțiune derulată pe 27 imagini !
Începe primul ciclu FETCH : se citește codul DD_H



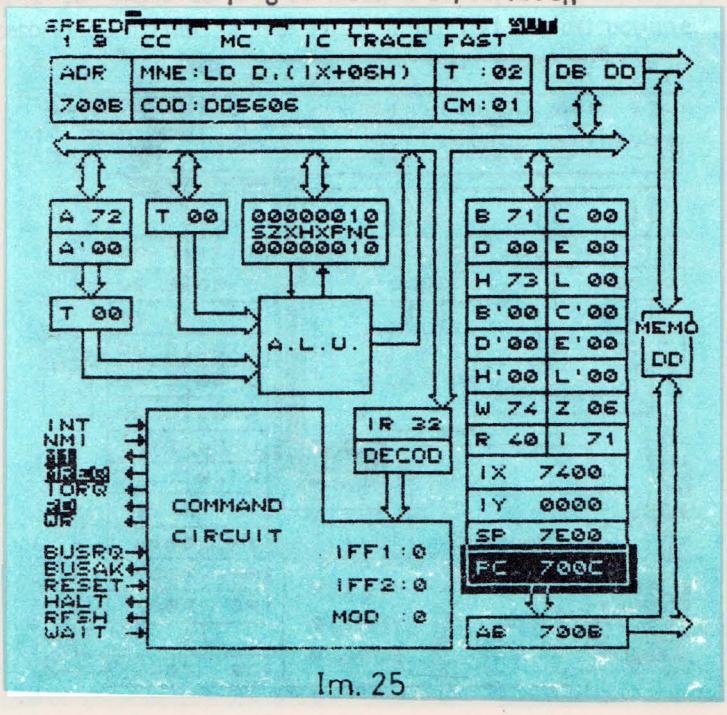
Im.23

Codul citit de la adresa $700B_H$ se depune în bufferul de date al microprocesorului



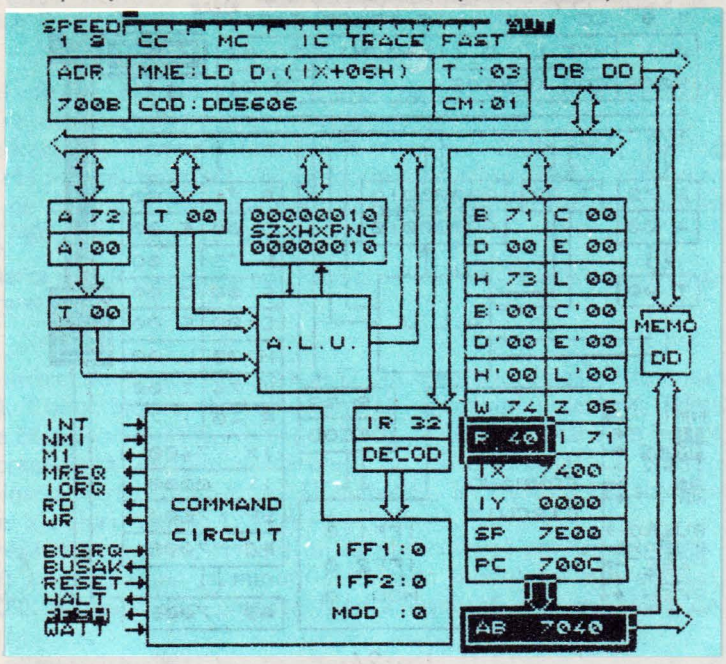
Im. 24

Se incrementează contorul program : $PC=PC+1=700C_H$



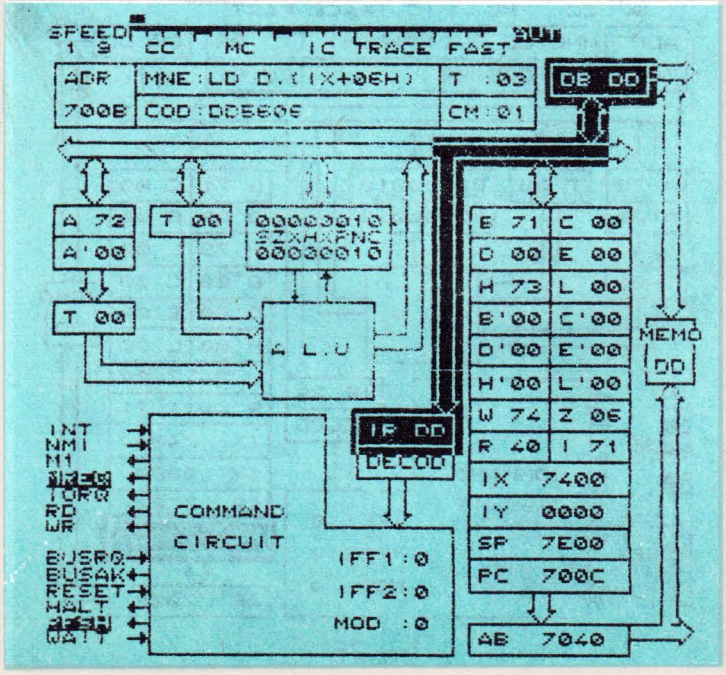
Im. 25

Conținutul registrului R se depune pe magistrala de adrese **A0 ÷ A6** și începe ciclul de reîmprospătare a memoriei dinamice (**$\overline{\text{RFSH}}$**—activ).



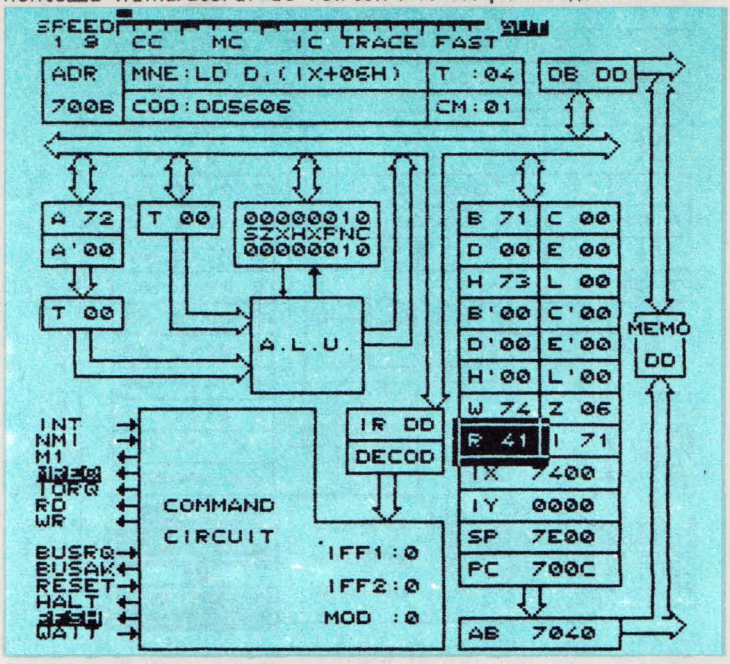
Im. 26

Concomitent codul citit DD_H se depune în registrul instrucțiune, pentru decodificare.



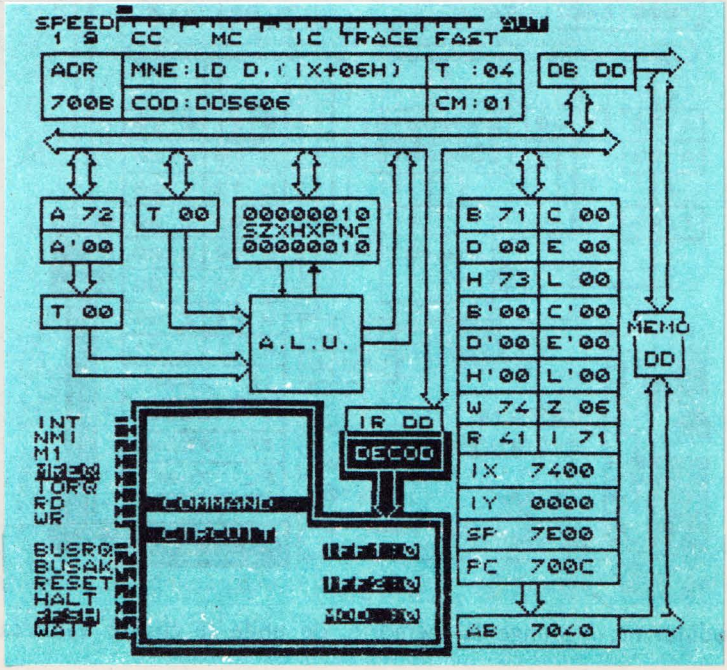
Im. 27

Se incrementează numărătorul de refresh : $R=R+1=41_H$.

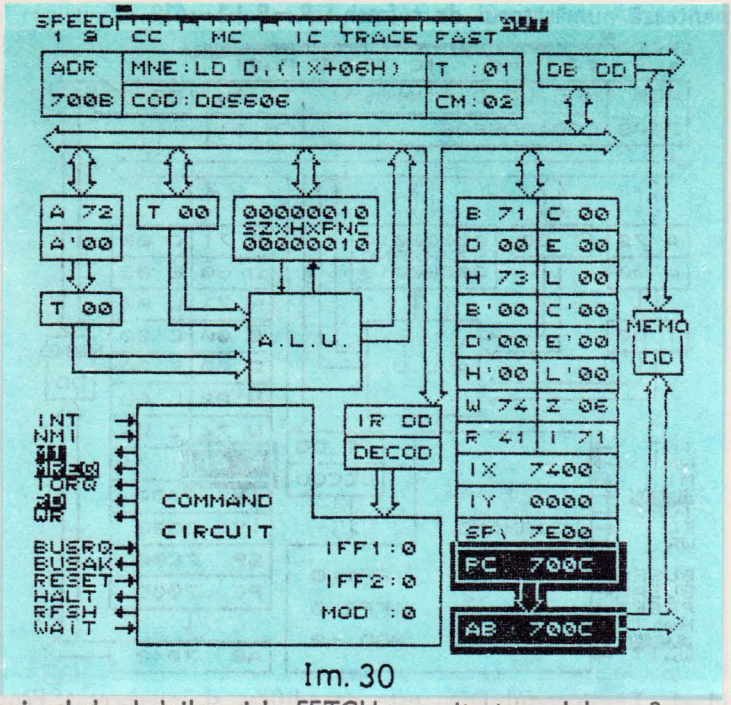


Im. 28

Acum (CM=1, T=4) codul DD_H este decodificat și interpretat. Procesorul „a înțeles" faptul că urmează o instrucțiune în care este implicat registrul index **IX.** Pentru a ști exact ce are de făcut, a cunoaște instrucțiunea propriu-zisă, microprocesorul va trebui să mai citească un cod de instrucțiune.

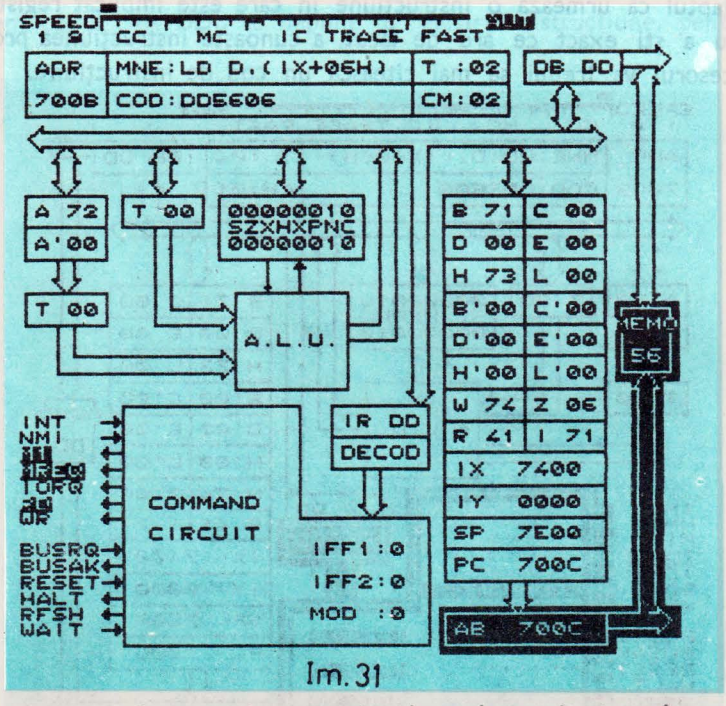


Im. 29



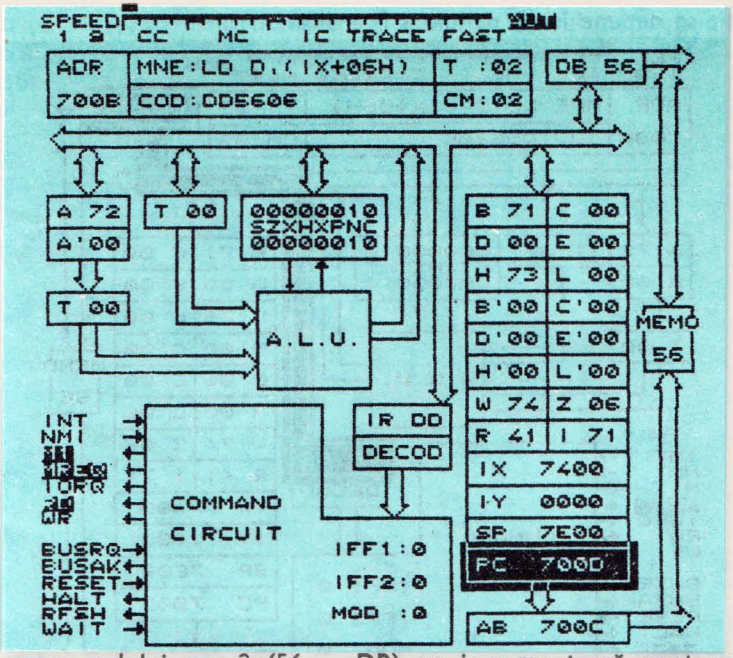
Im. 30

Începe deci cel de-al doilea ciclu FETCH : se citește codul nr. 2



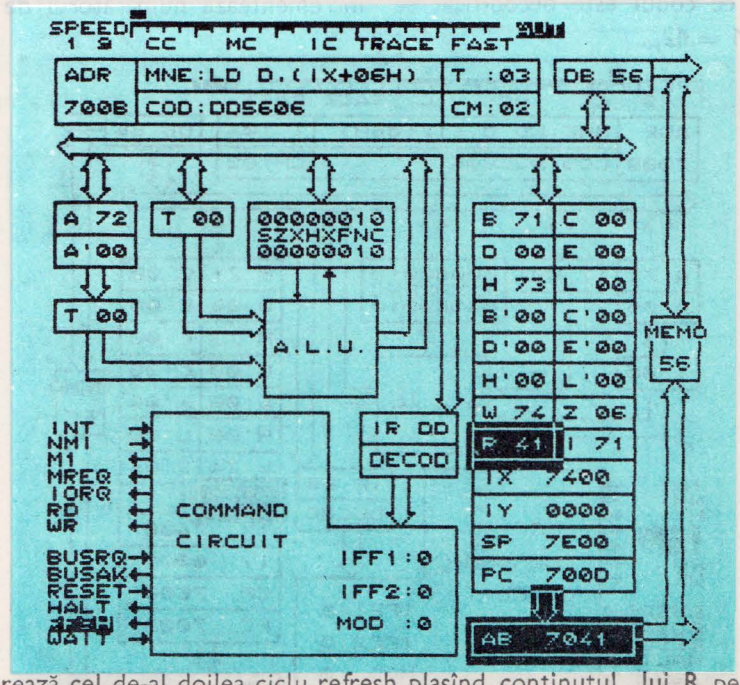
Im. 31

Adresa codului nr. 2 ajunge la memorie, de unde se citește valoarea 56_H.



Im. 32

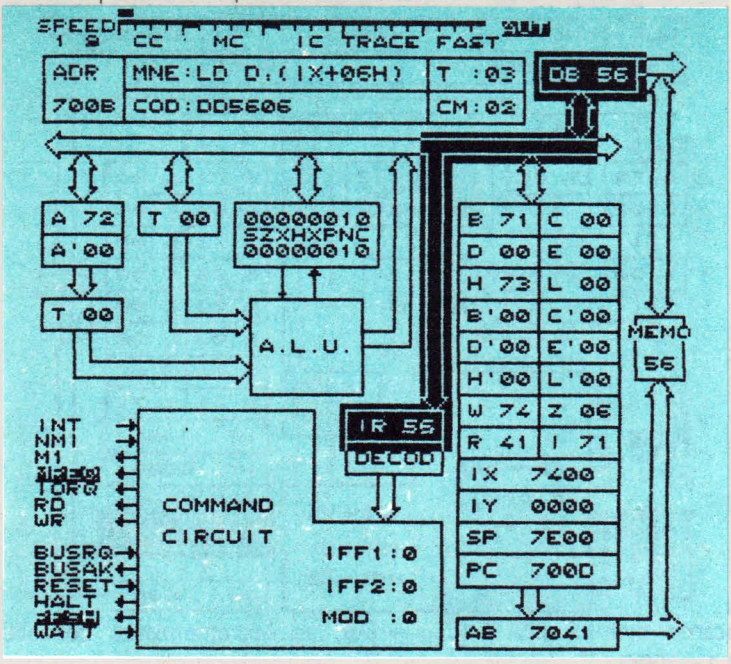
După stocarea codului nr. 2 ($56_H \rightarrow DB$) se incrementează contorul program $PC=PC+1=700D_H$.



Im. 33

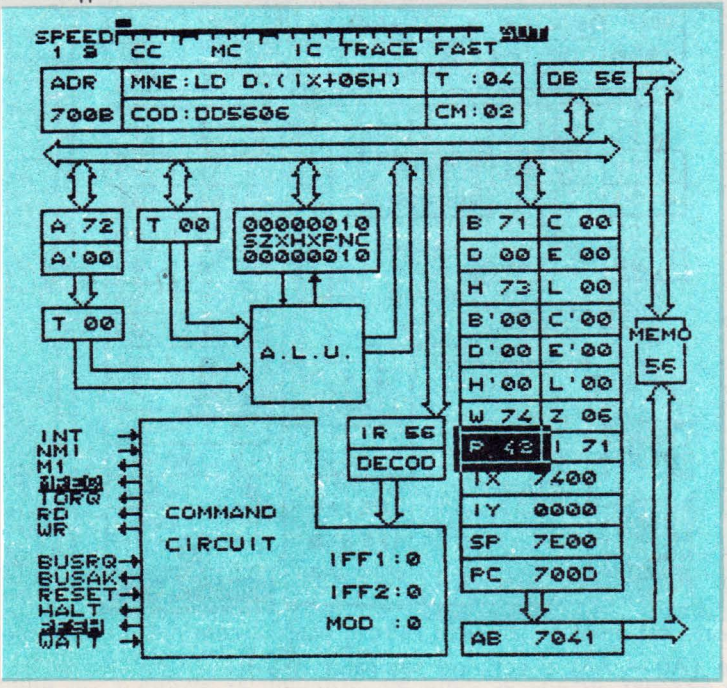
Se demarează cel de-al doilea ciclu refresh plasînd conţinutul lui R pe magistrala de adrese (A0 — A6) şi activînd semnalul $\overline{RFSH}$.

Codul citit se depune în IR pentru a fi analizat.



Im. 34

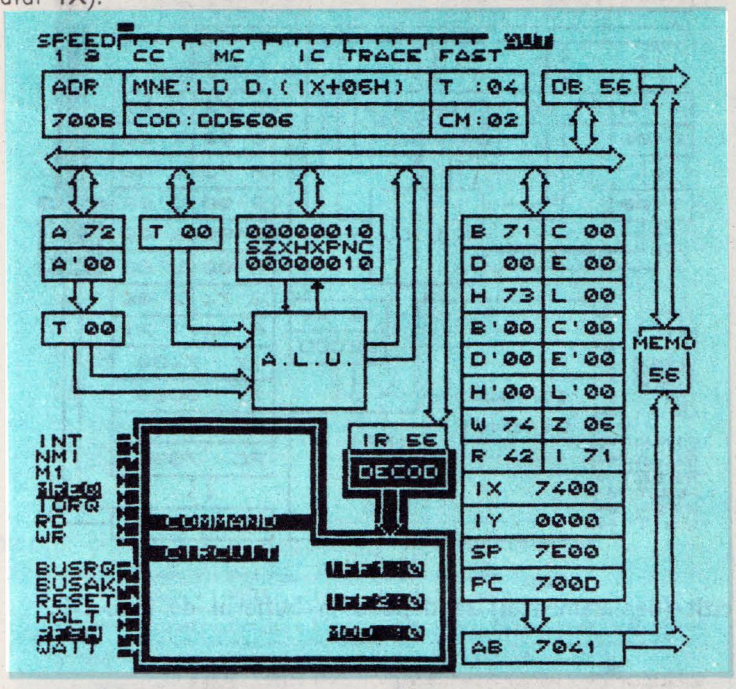
În timp ce codul este decodificat se incrementează numărătorul de refresh: $R = R+1 = 42_H$.



Im. 35

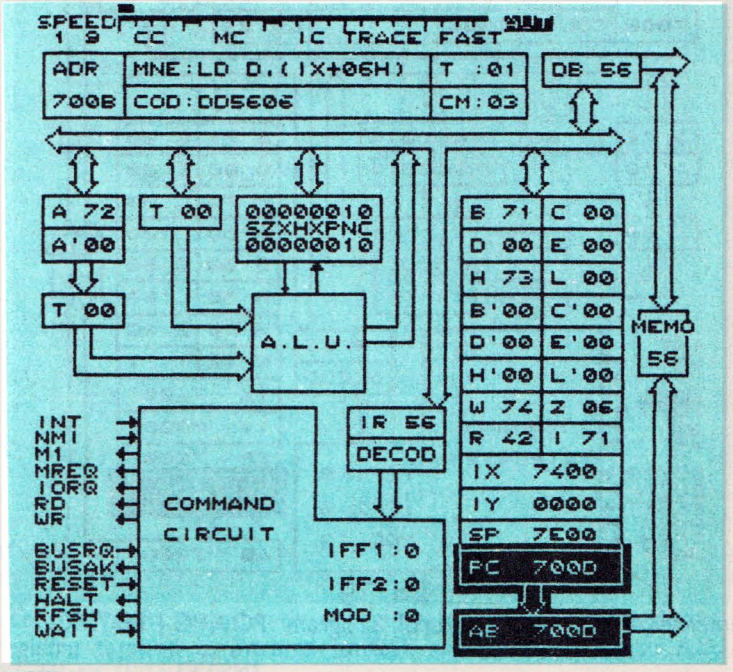
Abia acum (CM=2,T=4) microprocesorul a înțeles sensul instrucțiunii primite : trebuie încărcat registrul **D** cu conținutul unei celule de memorie.
Trebuie citit indicele variabilei căutate (deplasamentul față de valoarea curentă a registrului **IX**).

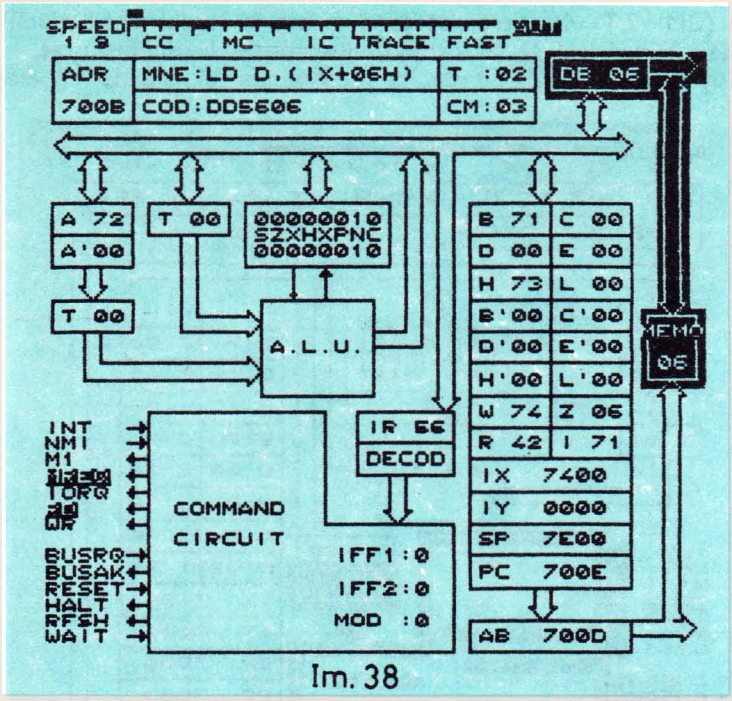
Im. 36



Începe un ciclu de citire a memoriei (READ). **PC** se depune pe magistrala de adrese.

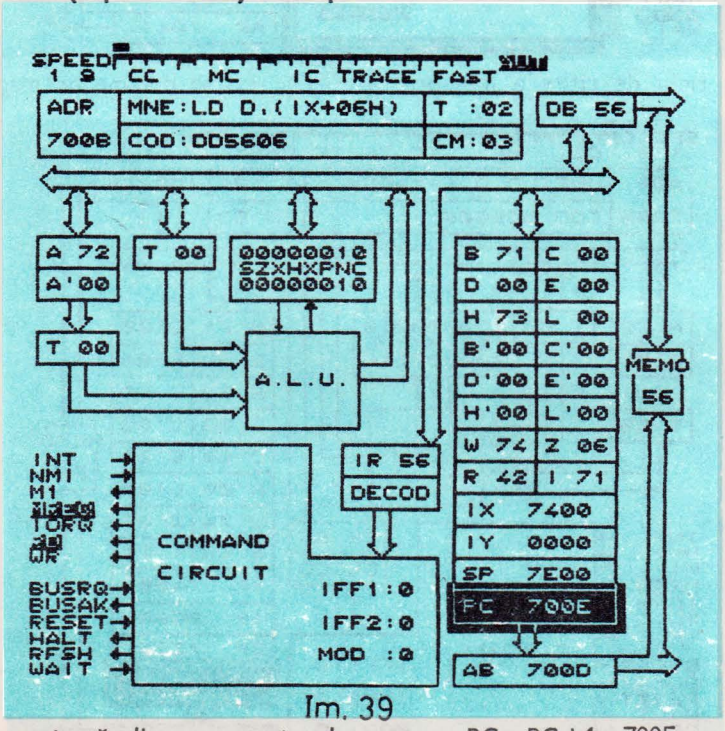
Im. 37





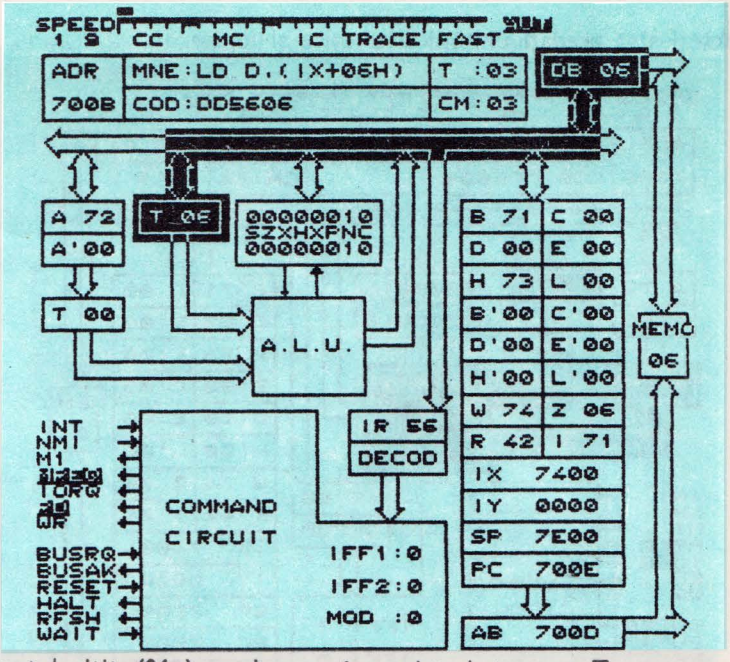
Im. 38

Octetul citit (deplasamentul) se depune în bufferul de date **DB**,



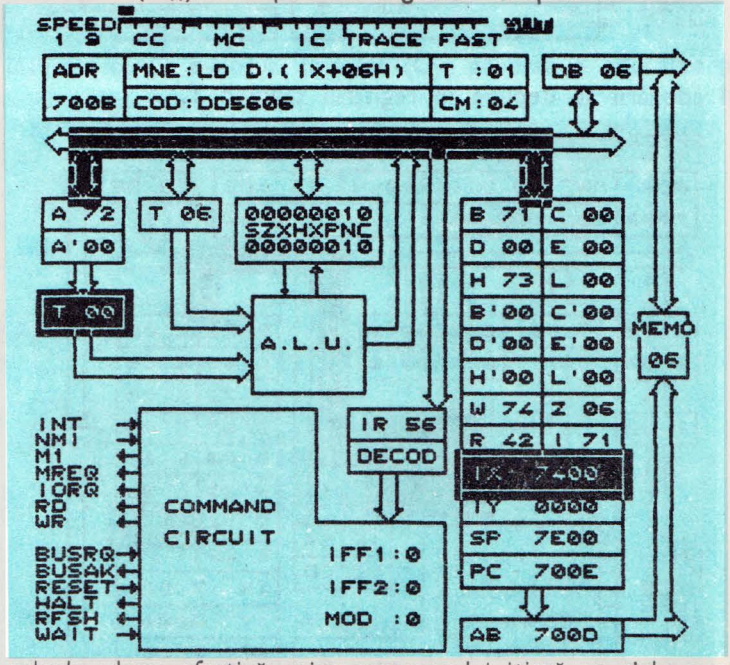
Im. 39

Se incrementează din nou contorul program $PC=PC+1=700E_H$.



Im. 40

Deplasamentul citit (06_H) se depune în registrul tampon T.

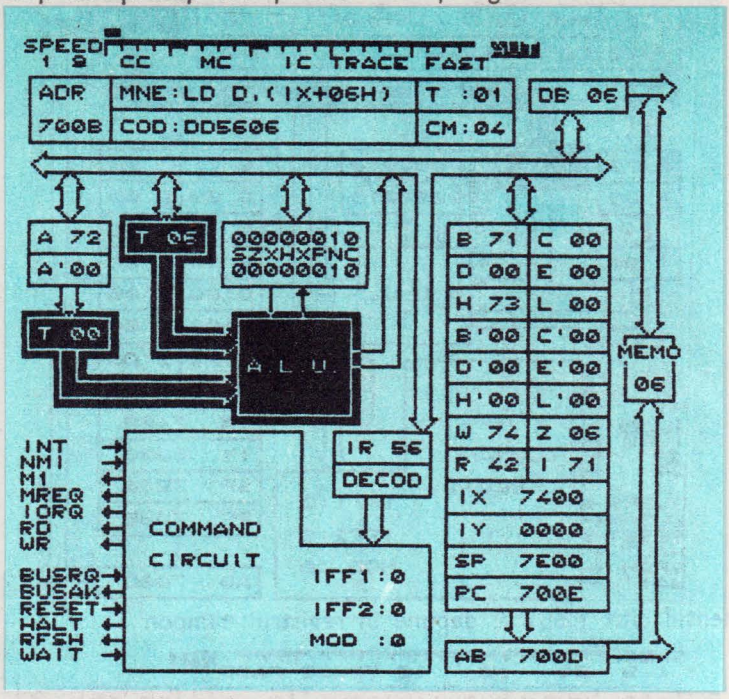


Im.41

Pentru a calcula adresa efectivă, microprocesorul inițiază un ciclu mașină intern, pe parcursul căruia toate semnalele externe vor fi inactive. Atenție deci : vom vedea acum un ciclu mașină pe care nici osciloscopul, nici analizorul de stări logice nu ni-l poate dezveli. Acesta este privilegiul lui **VISIBLE—Z80.**

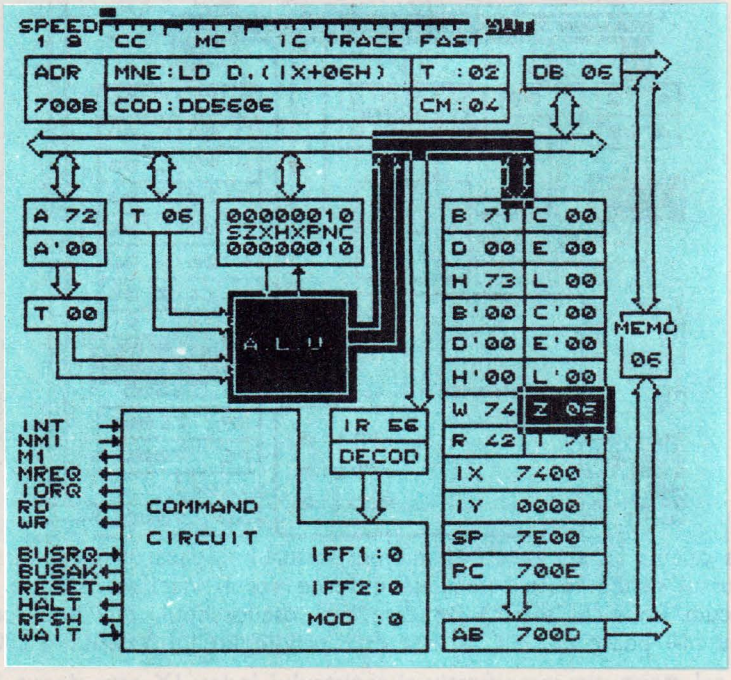
Octetul cel mai puțin semnificativ al registrului index **IX** este depus în cel de-al doilea registru tampon la intrarea unității aritmetice și logice **ALU.**

Cei doi octeți sînt predați unității aritmetice și logice.



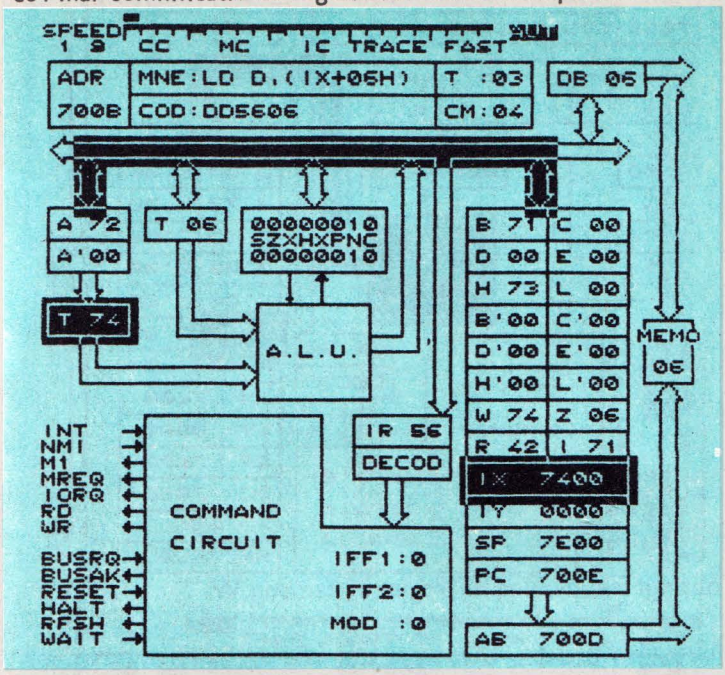
Im. 42

Rezultatul adunării se depune în registrul tampon Z.



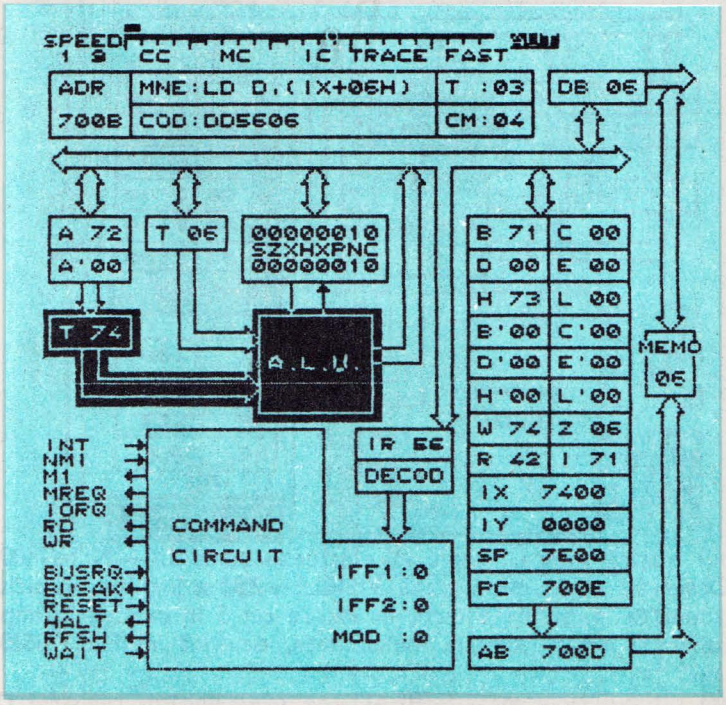
Im. 43

Octetul cel mai semnificativ al registrului **X** se transportă către **ALU**,

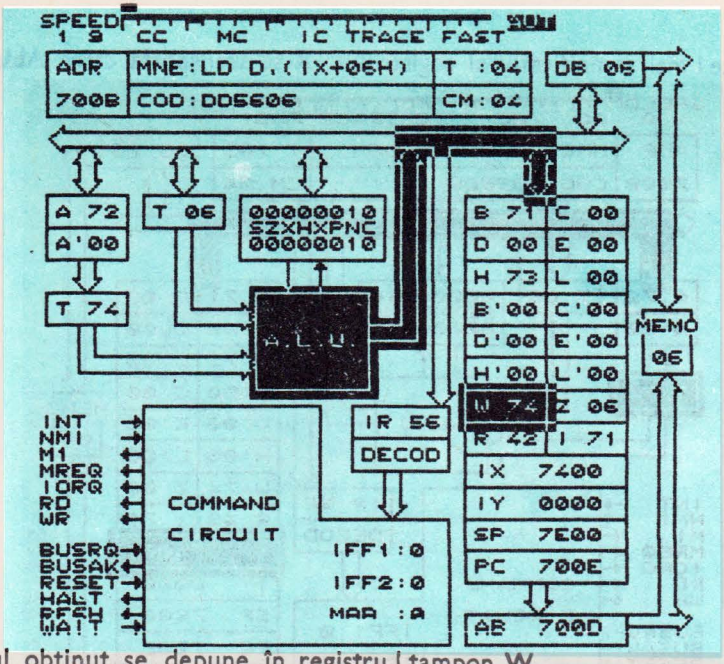


Im. 44

spre a fi însumat cu un eventual transfer -(Cy) generat la adunarea precedentă

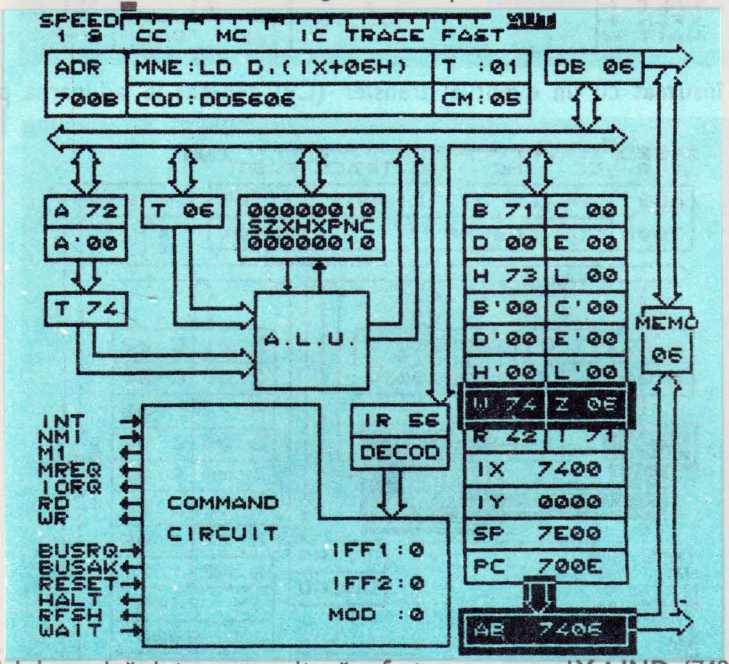


Im. 45



Im.46

Rezultatul obţinut se depune în registru l tampon **W**.

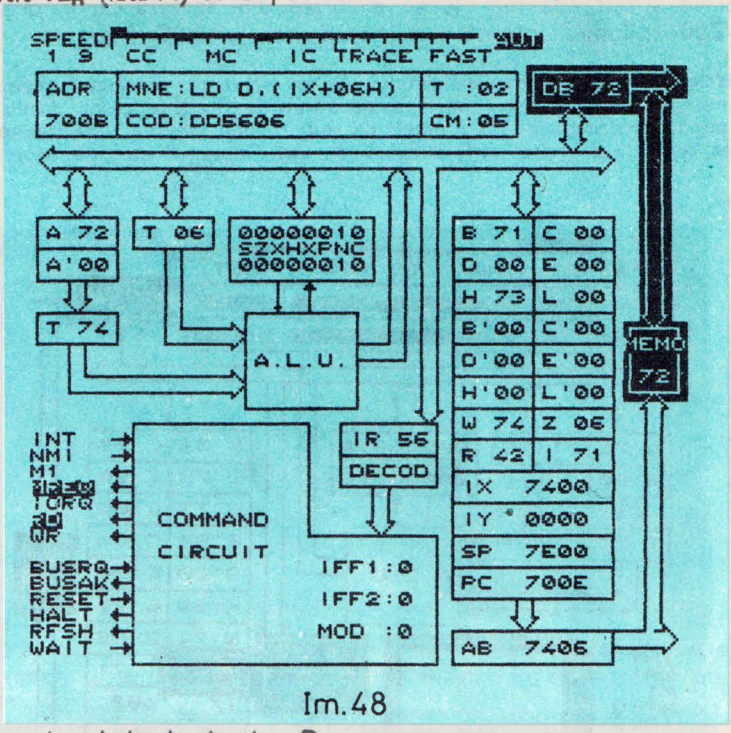


Im.47

Astfel ciclul maşină intern, menit să efectueze suma **IX+IND** (7400+06) s-a terminat. Observăm că în mare grabă **VISIBLE—Z80** a înghiţit un ciclu de tact, efectuînd adunarea în numai 4 tacţi, în loc de cei 5 în care se întîmplă în realitate. Propunem să iertaţi această inexactitate, de vină nefiind **VISIBLE—Z80**, ci autorii săi.

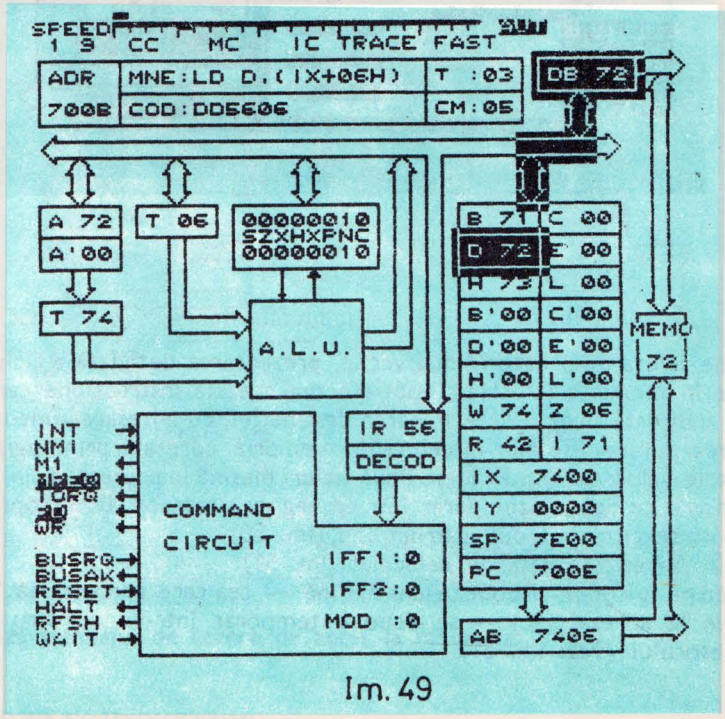
Începem ultimul ciclu maşină (READ) pentru a încărca registrul **D**. Adresa calculată se depune din **WZ** pe magistrala de adrese.

Octetul citit 72_H (iată-l !) se depune în bufferul de date



Im.48

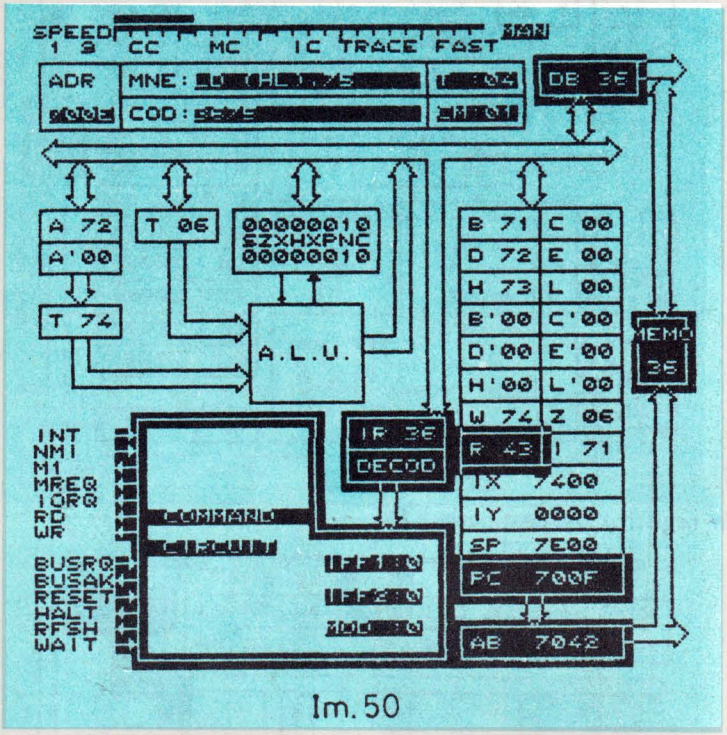
și apoi în registrul de destinație : D.



Im.49

Astfel ia sfîrşit cea mai lungă excursie din această carte, organizată cu **VISIBLE—Z80** în lumea internă a microprocesorului **Z80**. Reţinem:

Remarca 9: Însumarea deplasamentului de un octet cu conţinutul regiştrilor index IX sau IY, necesită un ciclu maşină intern avînd durata de 5 tacţi procesor. Acolo unde acest lucru este posibil (instrucţiunile cu 3 octeţi de cod) ciclului intern îi este suprapus un ciclu fetch, citirea celui de-al treilea cod, cîştigîndu-se astfel timp în execuţia unor instrucţiuni indexate.



Im. 50

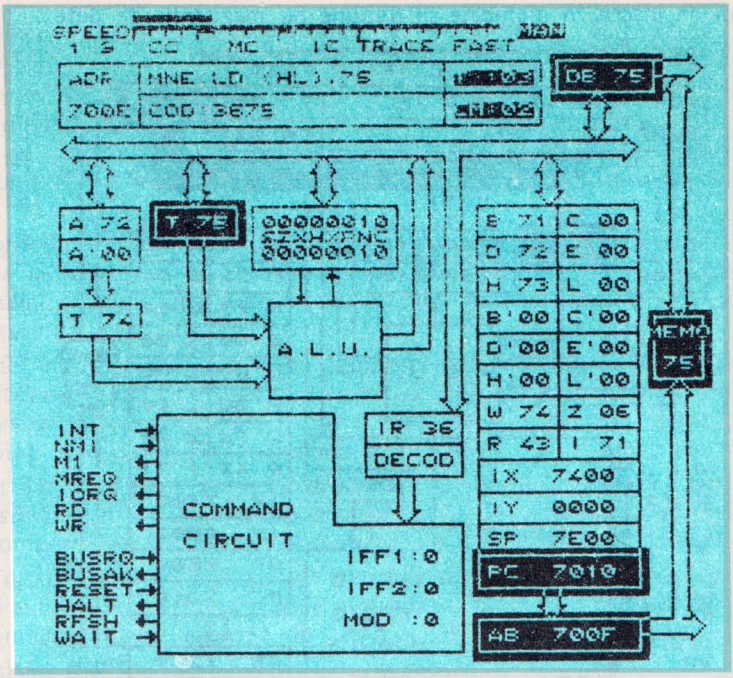
700E LD (HL),75 ; Im.50—Im.54 (MC şi CC)

Pentru a ne achita de o datorie mai veche, prezentarea detailată a ciclului maşină de scriere în memorie (WRITE), vom efectua această instrucţiune care combină adresarea indirectă pentru determinarea destinaţiei, cu adresarea imediată pentru identificarea sursei. Efectul: în celula de memorie adresată prin registrul dublu **HL** se înscrie valoarea 75_H. Primele două cicluri maşină le prezentăm în modul **MC**. Data de înscris se depune temporar într-un registru tampon al microprocesorului.

MC=1 : se citeşte codul instrucţiunii (Im. 50)

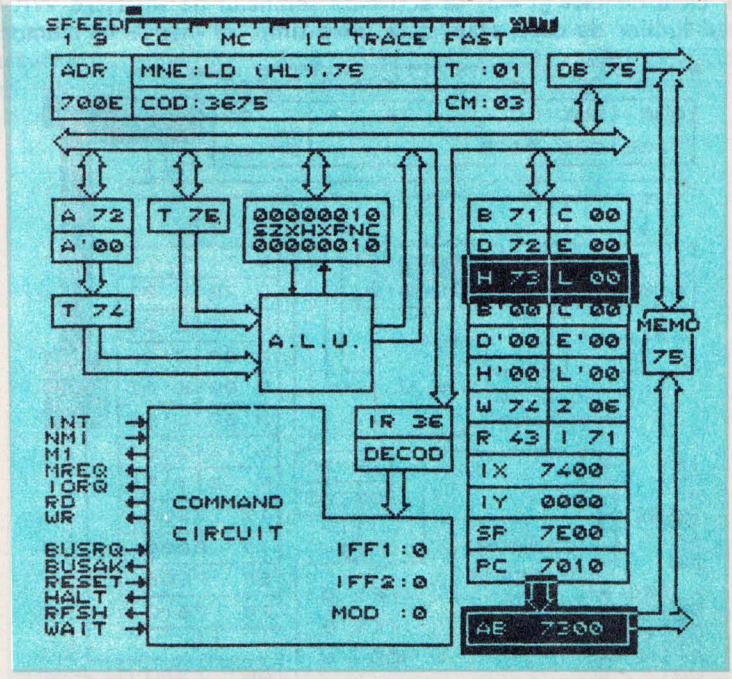
MC=2 : se citeşte data de vehiculat (Im. 51)

Data citită în cel de-al doilea ciclu maşină — cea care urmează să fie înscrisă în memorie la adresa (HL) — se depune temporar într-un registru tampon al microprocesorului (vezi Im. 51).



Im.51

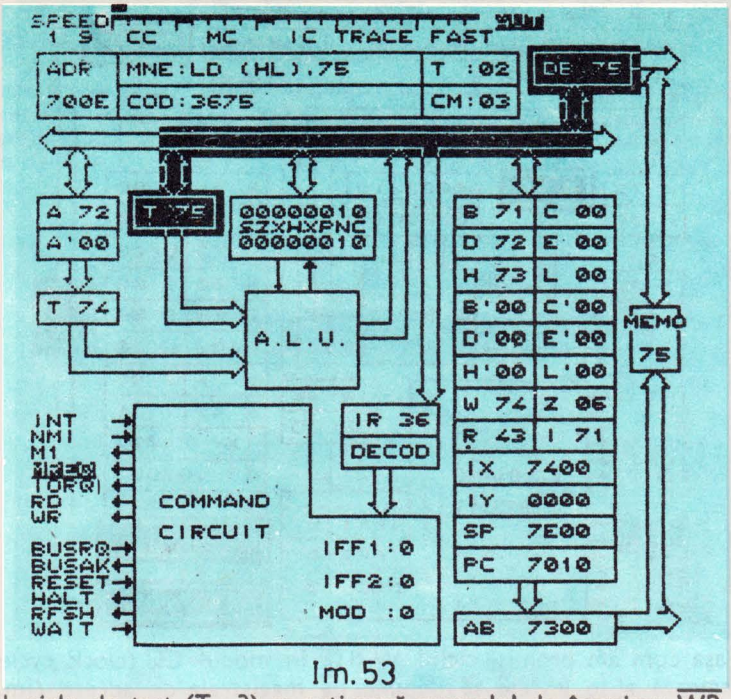
Urmează așa cum am promis, ciclul WRITE în modul CC (clock cycle). Adresa la care urmează a se înscrie se depune pe magistrala de adrese. (Im. 52)



Im.52

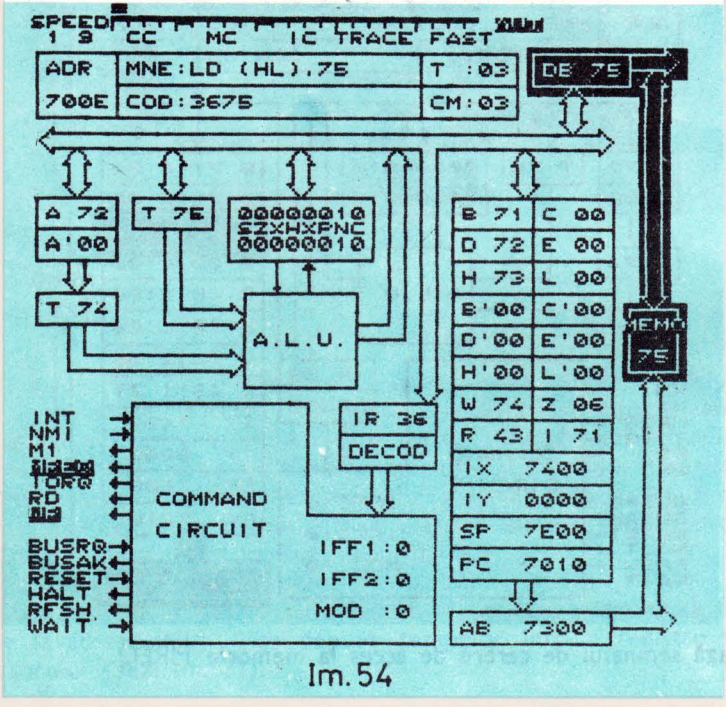
Se activează semnalul de cerere de acces la memorie $\overline{\text{MREQ}}$.

Apoi octetul de înscris în memorie se depune pe magistrala de date:



Im.53

În ultimul ciclu de tact (T=3) se activează semnalul de înscriere $\overline{WR}$ (WRITE), și conținutul liniilor de date, stabilizate între timp, se înscrie în memorie.



Im.54

Iată deci o instrucțiune care se descrie pe 2 octeți și se execută în trei cicluri mașină.

Amintim faptul că ciclul de scriere în memorie durează fără stări WAIT, 3 cicluri de tact. În cazul utilizării unor circuite de memorie mai *lente*, electronica lor de comandă va trebui să ativeze semnalul WAIT, prin care i se cere procesorului să mai aștepte, căci ciclul de memorie nu s-a terminat pînă în acest moment.

În acest din urmă caz microprocesorul va insera un ciclu de tact suplimentar, urmînd ca la jumătatea lui, să examineze din nou linia WAIT. În cazul în care aceasta indică în continuare cerere de așteptare, microprocesorul va insera un nou ciclu de tact de așteptare.

Astfel numărul de cicluri de tact aferenți unui ciclu de memorie (fetch, read, write) sau a unui ciclu de intrare/ieșire (in/out) este variabil. În exemplele noastre prezentate pînă în acest punct al cărții de față, am considerat toate ciclurile ca fiind executate fără cicluri (stări) WAIT.

Am rulat deci toate secvențele la viteza maximă, exploatînd astfel în totalitate posibilitățile fizice ale microprocesorului. Către sfîrșitul primei părți vom detaila comportamentul microprocesorului, în cazul utilizării semnalului WAIT.

Am ajuns astfel la sfîrșitul primei noastre aventuri, călătorie pe parcursul căreia am avut deja prilejul de a vede majoritatea blocurilor funcționale ale modelului de microprocesor generat de către VISIBLE — Z80, în acțiune.

De altfel toată această pagină este o varză.

Cele neutilizate în cadrul clasei LOAD—8, vor fi utilizate în aventurile menite să-l inițieze pe cititor în tainele celorlalte clase de instrucțiuni.

Încheiem în acest punct, nu înainte de a vă aminti faptul că modelul grafic de structură internă, generat de către VISIBLE—Z80, este unul teoretic și restrîns, el neputînd cuprinde într-o imagine de 256*256 puncte toate detaliile funcționale ale unui microprocesor real.

Desprindem o ultimă remarcă :

Remarca 10 : Întrucît incrementarea contorului program PC se face fără implicarea ALU şi are loc într-un singur tact procesor, rezultă că în interiorul microprocesorului trebuie să existe un modul funcţional care realizează această operaţie. În modelul grafic realizat cu VISIBLE—Z80 dispozitivul de incrementare / decrementare n-a fost figurat, din lipsă de spaţiu, dar el există şi afectează toţi regiştri.

Pentru a fixa mai bine concluziile primei aventuri şi pentru a lămuri eventualele nelămuriri ale cititorului, sintetizăm, mai riguros, tipurile de instrucţiuni de transfer pe 8 bit.

5.2. Sinteza clasei LOAD—8

Cele 111 instrucţiuni de transfer pe 8 bit ale microprocesorului **Z80** posedă aceeaşi mnemonică : **LD**.

Din punct de vedere funcţional le-am împărţit în 11 grupe. Cu excepţia instrucţiunilor **LD A, I** şi **LD A, R**, instrucţiunile de transfer pe 8 bit nu afectează indicatorii de condiţie.

Instrucţiunile de citire a regiştrilor hardware **LD A, I** şi **LD A, R** poziţionează flagurile de semn şi zero, copiind totodată conţinutul bistabilului de întrerupere IFF, în indicatorul P/V.

Filele grupelor de instrucţiuni din clasa LOAD—8 se regăsesc în partea a II-a COMPLEMENTE, § E.2.

5.3. Aventura a 2-a : transferuri de 16 bit

„Lăsaţi la vatră", să începem o nouă aventură, tot de la adresa 7000_H, adresă pe care v-o sugerăm şi d-voastră ca adresă de început pentru programele (nu mai mari de 4k) care doriţi să le executaţi cu **VISIBLE—Z80**. Itinerariul propus este :

```
7000    LD      IX,7654         ; Im. 55        (IC)
7004    LD      SP, IX          ; Im. 56        (IC)
7006    EX      (SP), IX        ; Im. 57—Im. 62 (MC)
7008    LD      HL, (7654)      ; Im. 63        (IC)
700B    EX      DE, HL          ; Im. 64        (IC)
700C    POP     HL              ; Im. 65        (IC)
700D    PUSH    DE              ; Im. 66        (IC)
700E    LD      BC, (7654)      ; Im. 67        (IC)
```

Secvenţa, deşi scurtă, atinge toate cele patru mnemonici ale clasei : **LD, EX, POP, PUSH**. Totodată vom putea trece în revistă şi tehnicile de adresare : imediată, implicită şi indirectă, folosind atît transferurile interregiştri cît şi cele între regiştri dubli şi cîte două locaţii de memorie adiacente.

Pentru ca secvența de sus să nu se execute chiar fără un sens intrinsec, vom începe din nou cu regiștri șterși (conținut 00H), urmînd ca la sfîrșitul aventurii majoritatea lor să conțină aceeași valoare : 7654_H.

Viteza de execuție va fi mai mare (IC — mod ciclu instrucțiune). Astfel atenția noastră se va redistribui : de la *„cum"* se poate face la *„ce"* se poate face.

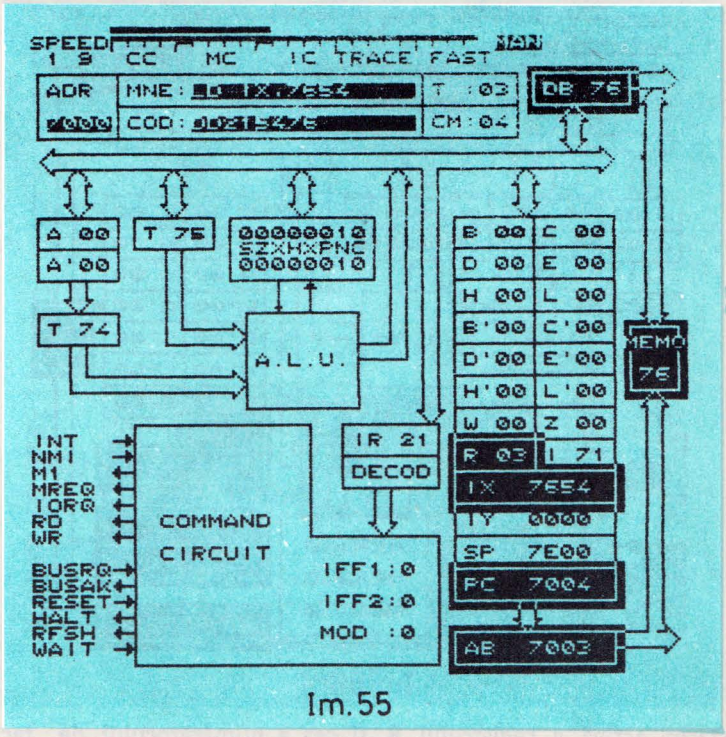
```
7000    LD    IX,7654        ; Im. 55        (IC)
```

Instrucțiunea execută încărcarea imediată a registrului dublu **IX** cu valoarea 7654_H.

Instrucțiunea ocupă patru octeți în memorie : $DD_H21_H54_H76_H$.

Execuția durează 14 cicluri de tact repartizate pe 4 cicluri mașină: **4, 4, 3, 3.**

Primul popas (Im. 55) îl facem la sfîrșitul acestei instrucțiuni.



Im.55

Remarca 11 : Toți regiștri dubli pot fi implicați în instrucțiunile de transfer cu adresare imediată, aceste instrucțiuni dovedindu-se a fi foarte democratice.
Observăm că ultimii 3 octeți ai instrucțiunii prezentate ($21_H54_H76_H$), reprezintă de fapt instrucțiunea : **LD HL,7654** adică o instrucțiune cu efect similar celei precedente, schimbîndu-se doar registrul de destinație al transferului (din **IX** în **HL**). Observația permite formularea unei reguli generale.

Remarca 12 : Din punct de vedere constructiv regiștri dubli de index **IX** și **IY** sînt asociați registrului dublu de uz general **HL**.

Instrucţiunile ce operează cu regiştri de index vor avea cîte un cod rezervat (DD_H sau FD_H) pentru identificarea registrului index (IX sau IY), urmat de un cod efectiv (cel care specifică de fapt instrucţiunea) identic cu codul instrucţiunii care efectuează o operaţie similară, dar cu registrul HL. Datorită faptului că registrul dublu HL este oarecum privilegiat faţă de ceilalţi doi (BC şi DE) — avînd mai multe posibilităţi de adresare — şi regiştri index IX, IY se vor scălda în lumina fratelui lor mai mare HL.

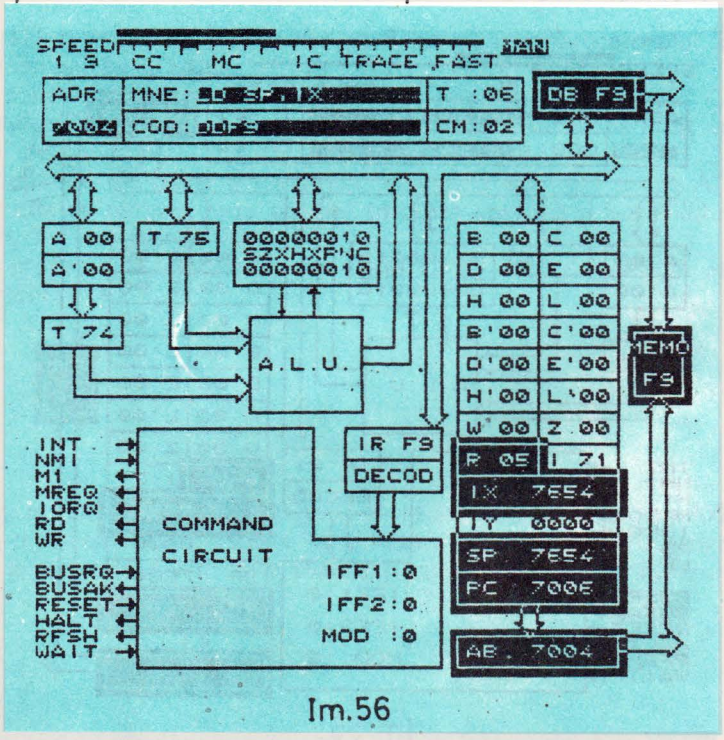
Aşa cum se va vedea pe următorul traseu al aventurii noastre :

```
7004    LD      SP,IX          ; Im. 56        (IC)
```

Instrucţiunea încarcă registrul indicator de stivă SP, cu valoarea conţinută în registrul dublu IX.

Codul instrucţiunii ocupă 2 octeţi : DD_HF9_H (ar putea cineva să ne spună codul instrucţiunii LD SP, HL ?)

Execuţia durează 10 cicluri de tact repartizate în două cicluri maşină : 4, 6



Im.56

Din starea activă a registrului R şi cea a numărătorului de tact (T = 6 deducem că cel de-al doilea ciclu maşină efectuat a fost un ciclu fetch extins

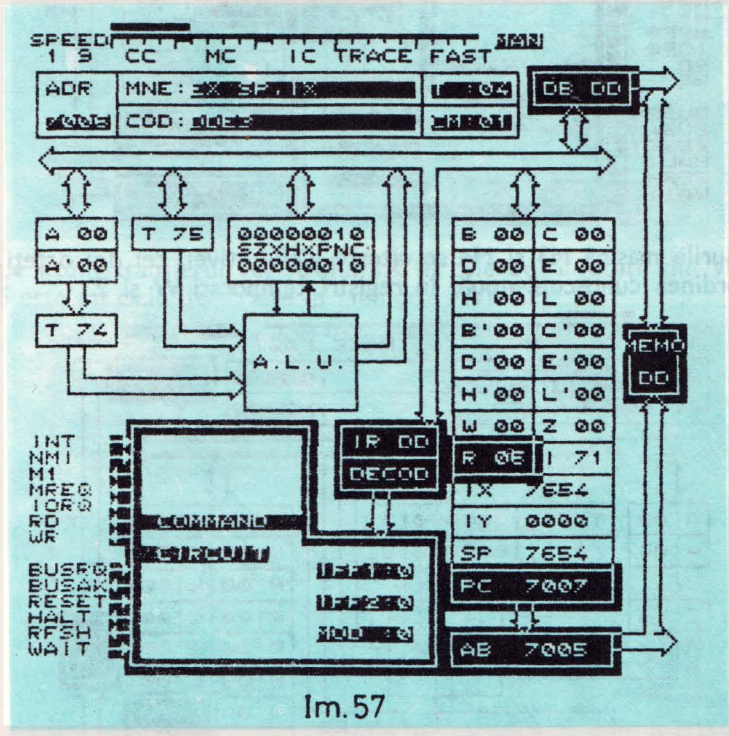
Remarca 13 : Încărcarea indicatorului de stivă din alt registru este posibilă doar dacă, ca sursă se foloseşte unul din regiştri „privilegiaţi" HL, IX sau IY.
Ex. : Nu există instrucţiunea LD SP, BC !
Astfel numărul modurilor de încărcare a indicatorului de stivă SP se ridică la 3 :
— încărcare imediată : LD SP, număr
— încărcare directă din memorie : LD SP, (adresă)
— încărcare din alt registru : LD SP, rr unde rr poate fi HL, IX sau IY.

Instrucţiunea interschimbă conţinutul registrului dublu **IX** cu cel al celor 2 locaţii de memorie adiacente, care se află în vîrful stivei (adresa vîrfului stivei se află în indicatorul de stivă **SP**).

Instrucţiunea ocupă 2 octeţi în memorie : DD_HE3_H.

Execuţia se execută în 6 cicluri maşină, însumînd 23 tacţi procesor : **4, 4, 3, 4, 3, 5.**

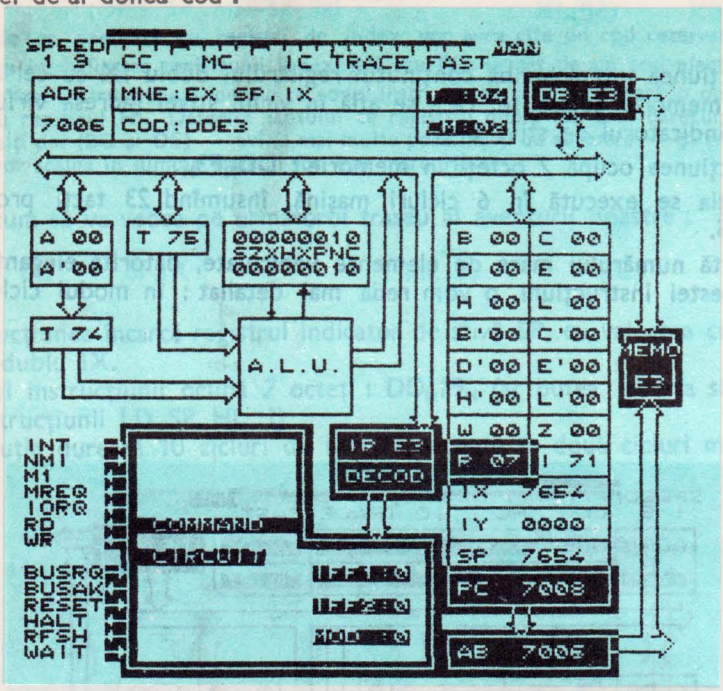
Datorită numărului mare de elemente vehiculate, datorită eleganţei şi frumuseţii acestei instrucţiuni, o vom reda mai detailat : în modul ciclu maşină.



Im. 57

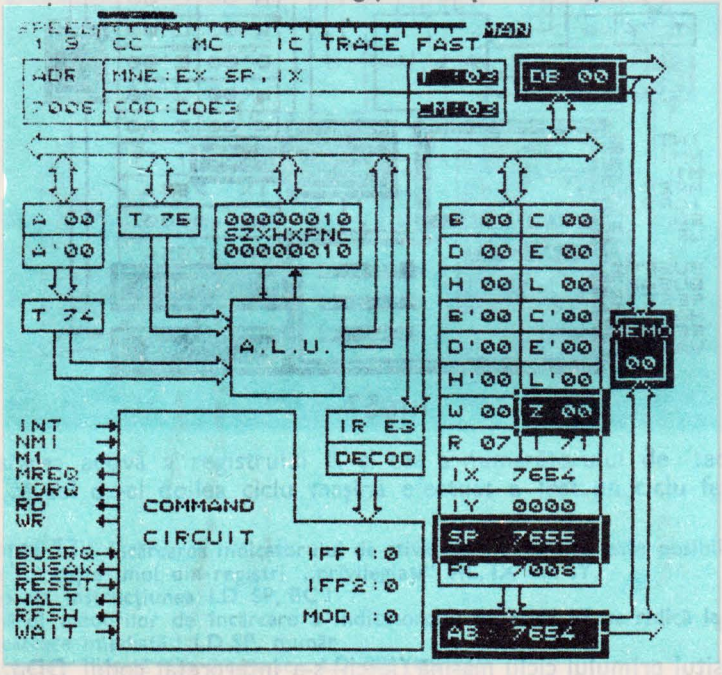
La sfîrşitul primului ciclu maşină (fetch) s-a interpretat codul DD_H.

Urmează cel de-al doilea cod :



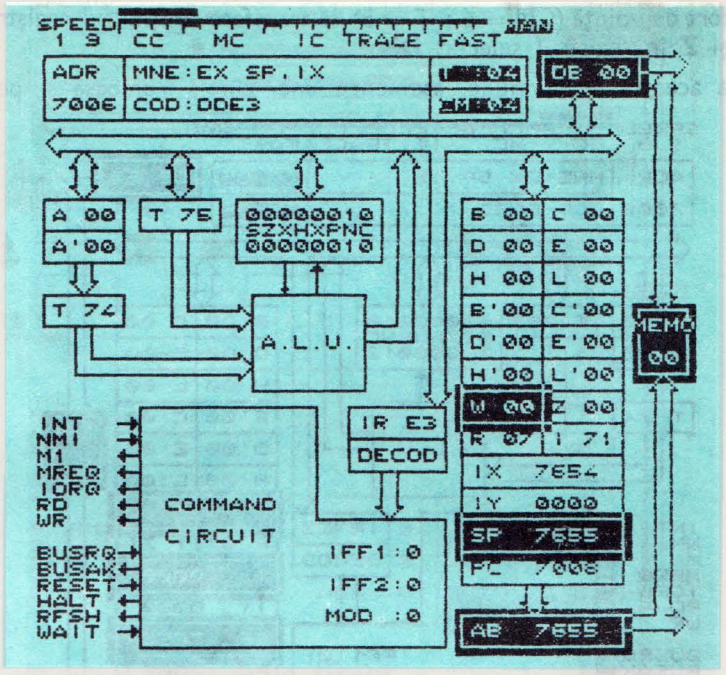
Im.58

În ciclurile mașină **M3** și **M4** se citește vîrful stivei, cei doi octeți depunîndu-se în ordinea cunoscută nouă, în regiștri temporari **W** și **Z**.



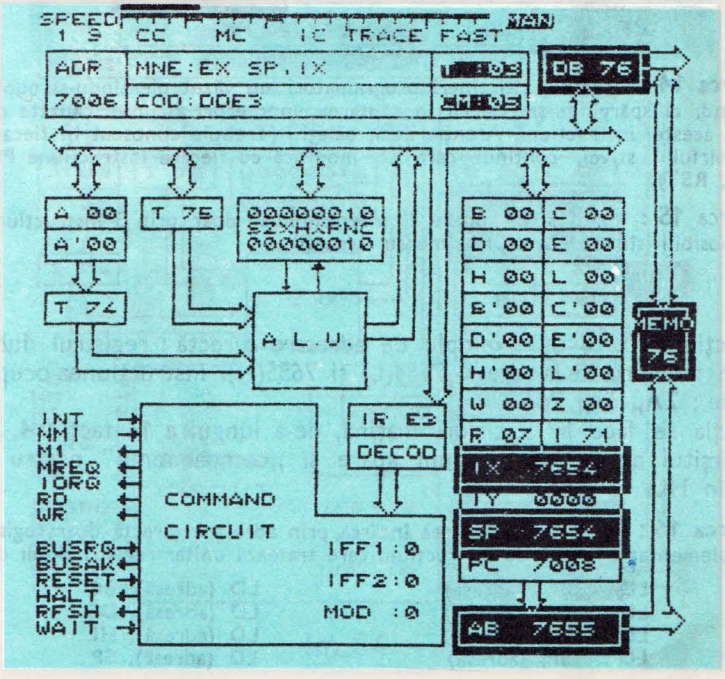
Im.59

Observăm că **SP** a fost incrementat pentru a citi cel de-al doilea octet.



Im. 60

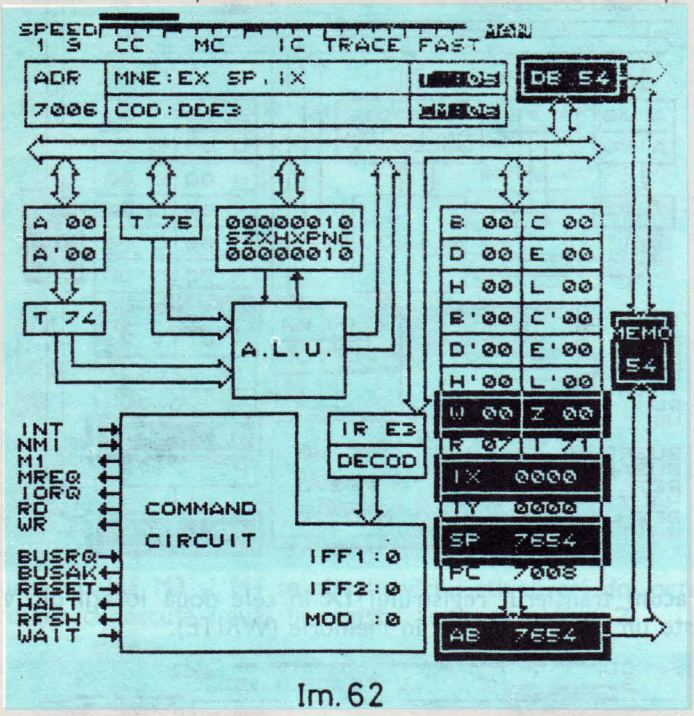
Începe acum transferul registrului **IX** în cele două locații din vîrful **stivei.** Ciclul **M5** este un ciclu de scriere în memorie (WRITE).



Im. 61

Cel de-al doilea ciclu WRITE salvează ultimul octet 54_H, pentru ca apoi un ultim efort de voință (CM = 6 ; T = 5) să transfere conținutul regiștrilor temporari **W** și **Z** în registrul selectat **IX**.

Feerică această instrucțiune, asemenea unei zîne : frumoasă și periculoasă.



Im.62

Remarca 14 : Doamne, cîți tineri programatori am văzut pierzîndu-și nopțile cu ea ! Dar nu visînd, ci spărgîndu-și creierii în căutarea unor erori ascunse, cauzate de folosirea prematură a acestei instrucțiuni. Atenție deci, băieți ! (Trebuie cunoscut în fiecare moment conținutul vîrfului stivei, conținut care se modifică cu fiecare instrucțiune **PUSH, POP CALL, RET, RST**).

Remarca 15 : Vîrful stivei poate fi interschimbat doar prin 3 instrucțiuni dedicate. Partenerii posibili sînt : **HL, IX, IY** (mereu aceeași).

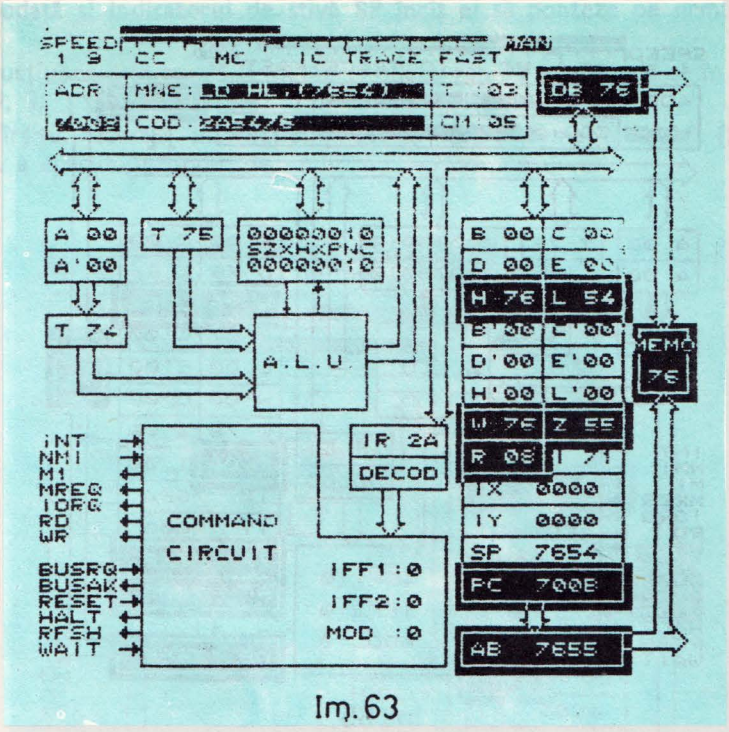
Instrucțiunea este un exemplu de adresare directă : registrul dublu **HL** se încarcă din memorie de la adresa 7654(L) și 7655(H). Instrucțiunea ocupă 3 octeți în memorie : $2A_H$ 54_H 76_H.

Execuția se face în 5 cicluri mașină, de-a lungul a 16 tacți : **4, 3, 3, 3, 3.**

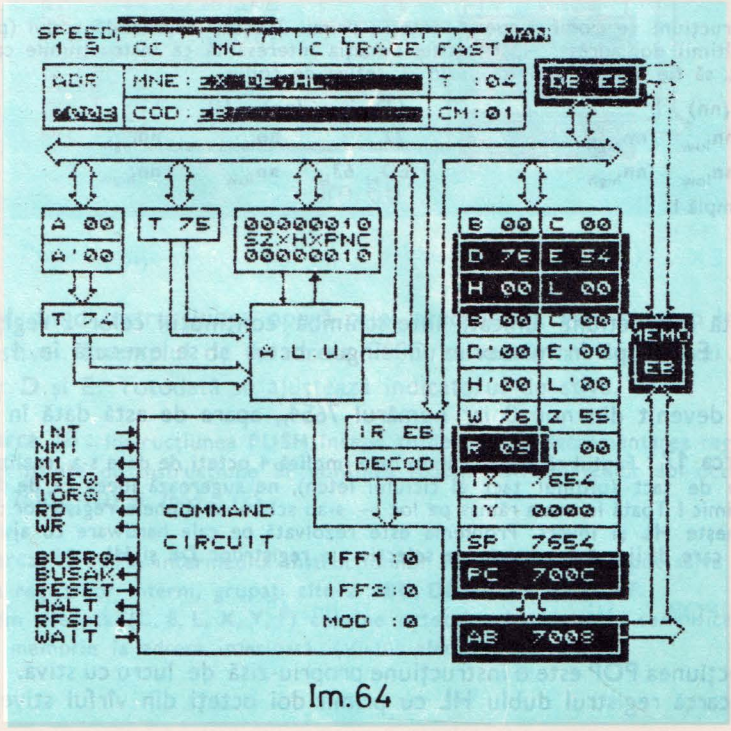
La sfîrșitul acestei instrucțiuni apare și ,,compagnonul" nostru 7654_H, de astă dată în **HL.**

Remarca 16 : Pe cînd **I8080** putea încărca prin adresare directă doar registrul **HL**, la **Z80** s-a implementat o grupă de instrucțiuni care tratează unitar regiștri dubli de bază :

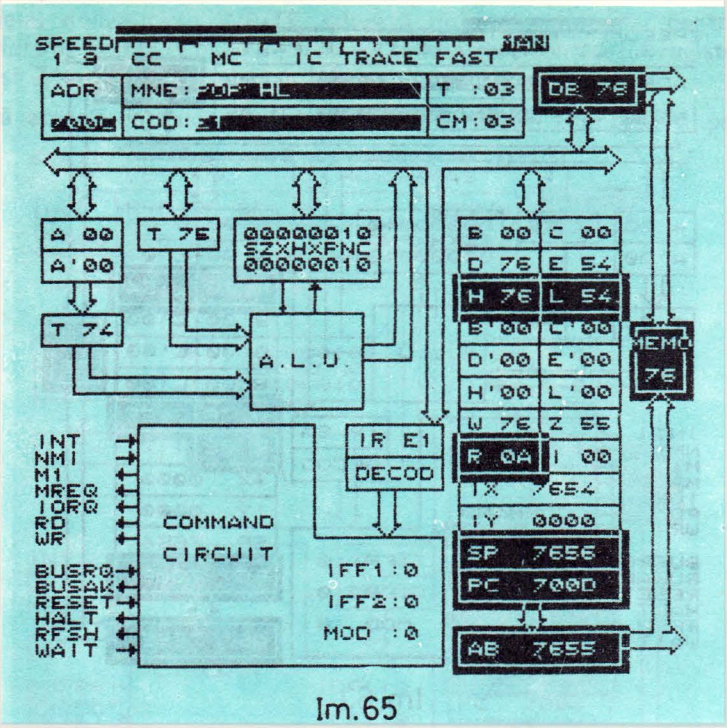
LD	BC, (adresa)	LD (adresă), BC
LD	DE, (adresă)	LD (adresă), DE
LD	HL, (adresă)	LD (adresă), HL
LD	SP, (adresă)	LD (adresă), SP


SPEED
1 S
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE:
T :03
DB 76
COD
CM: 05
A 00
A' 00
T 75
00000010
SZXHXPNC
00000010
B 00
C 00
D 00
E 00
H 76
L 54
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 76
Z 55
R 08
I 71
IX 0000
IY 0000
SP 7654
PC 700B
AB 7655
MEMO
76
T 74
A.L.U.
IR 2A
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.63


SPEED
1 S
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE:
T :04
DB EB
COD
CM:01
A 00
A' 00
T 75
00000010
SZXHXPNC
00000010
B 00
C 00
D 76
E 54
H 00
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 76
Z 55
R 09
I 00
IX 7654
IY 0000
SP 7654
PC 700C
AB 7009
MEMO
EB
T 74
A.L.U.
IR EB
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.64



Im.65

Aceste instrucțiuni se codifică pe 4 octeți : primii 2 octeți reprezintă codul (primul este **ED**), iar ultimii doi adresa. Apare astfel situația interesantă ca instrucțiunile care implică registrul **HL** să fie dublu definite, prin 2 seturi de coduri.

LD	**HL, (nn)**		**LD**	**(nn), HL**	
$2A_H$	nn_{low}	nn_{high}	22_H	nn_{low}	nn_{high}
$ED_H 68_H$	nn_{low}	nn_{high}	$ED_H 63_H$	nn_{low}	nn_{high}

Se mai întîmplă !

Această instrucțiune unicat, interschimbă conținutul celor *2* regiștri dubli specificați. Ea ocupă în memorie un singur octet și se execută în 4 cicluri de tact.

HL a devenit din nou 0 iar numărul 7654_H apare de astă dată în **DE**.

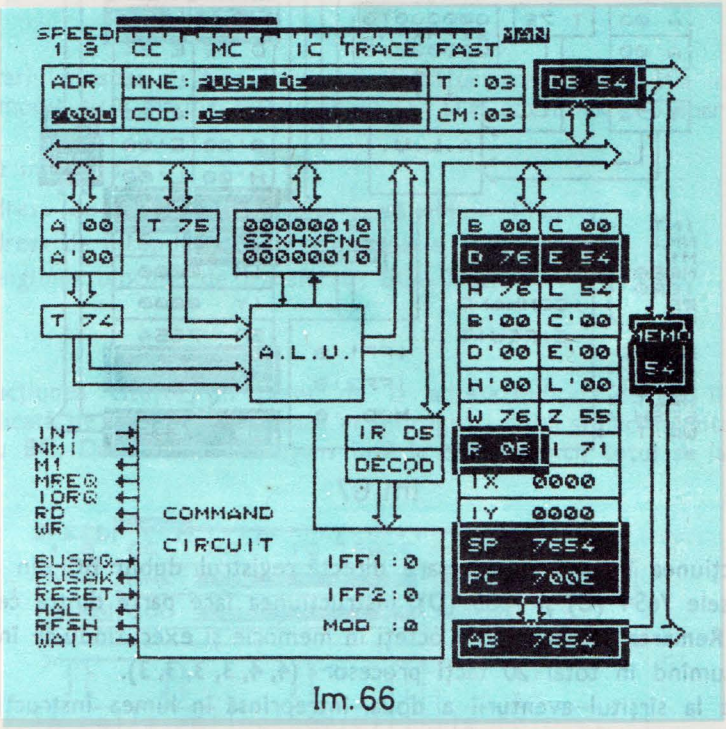
Remarca 17 : Faptul că acest schimb care implică 4 octeți de date s-a realizat într-un singur ciclu de tact (ultimul tact al ciclului fetch), ne sugerează ideea că de fapt nu s-a transferat nimic ! Toată lumea a rămas pe loc — s-au schimbat numele regiștrilor : ce a fost **DE** se numește **HL** și invers. Problema este rezolvată pe cale hardware cu ajutorul unui *multiplexor*, care dirijează semnalele de selecție ale regiștrilor **DE** și **HL**.

Instrucțiunea **POP** este o instrucțiune propriu-zisă de lucru cu stiva. Exemplul de față încarcă registrul dublu **HL** cu primii doi octeți din vîrful stivei, și ajus-

teză totodată şi indicatorul de stivă **SP** încît el să ponteze pe următorul grup de date.

Instrucţiunea este scurtă (1 octet) şi relativ rapidă, efectuîndu-se în trei cicluri maşină (4, 3, 3 tacţi procesor).

La sfîrşitul acestei instrucţiuni, **SP** a fost incrementat cu 2, ($SP=7656_H$) iar în **HL** a apărut de unde, de unde, nu -7654_H.



Im. 66

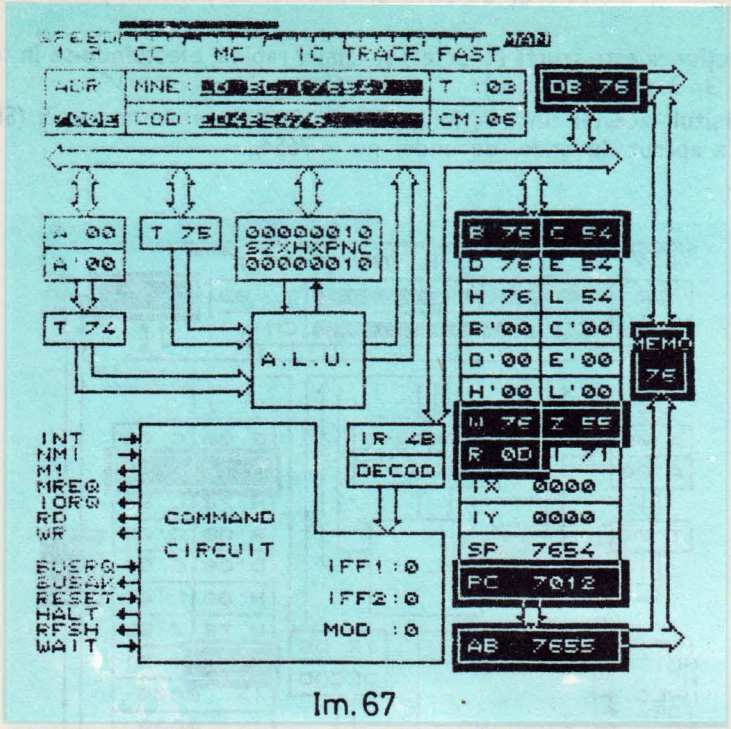
700D PUSH DE ; Im. 66 (IC)

PUSH, fiind instrucţiunea opusă celei precedente, este şi ea o instrucţiune dedicată stivei. Exemplul de la adresa 700D salvează în vîrful stivei conţinutul regiştrilor **D** şi **E**. Totodată se ajustează indicatorul de stivă.

Remarca 18 : Instrucţiunea **PUSH** începe totdeauna cu decrementarea registrului **SP**. Astfel dacă înainte de instrucţiune SP conţinea 7656_H, înscrierea celor doi octeţi s-a făcut la adresele 7655_H **(D)** şi 7654_H **(E)**.

Remarca 19 : Prin intermediul instrucţiunilor **PUSH** şi **POP** se pot salva şi restaura majoritatea regiştrilor interni, grupaţi cîte 2 : **BC, DE, HL, IX, IY** şi **AF**.
Registrul din dreapta **(C, E, L, X, Y, F)** conţine octetul cel mai puţin semnificativ, care se salvează în memorie la adresa inferioară. (Vîrful efectiv al stivei).

700E LD BC, (7654) ; Im. 67 (IC)



Im. 67

Instrucţiunea încarcă cu adresare directă registrul dublu **BC** din memorie, de la adresele 7654 **(C)** şi 7655 **(D)**. Instrucţiunea face parte dintre cele 8 prezentate la **Remarca 16**, ocupînd 4 octeţi în memorie şi executîndu-se în 6 cicluri maşină însumînd în total 20 tacţi procesor **(4, 4, 3, 3, 3, 3)**.

Iată-ne la sîrşitul aventurii a doua, întreprinsă în lumea instrucţiunilor de transfer pe 16 bit. (Acum şi **BC** conţine 7654_H !).

5.4. Sinteza clasei LOAD—16

Cu ajutorul a **5** mnemonici **(LD, EX, EXX, POP, PUSH)** se realizează **39** de instrucţiuni de transfer pe 16 bit, dintre care **33** unidirecţionale şi **6** bidirecţionale. Cu excepţia instrucţiunilor **EX AF, AF'**, şi **POP AF** care încarcă registrul **F**, nici una din instrucţiuni nu afectează indicatorii de condiţie.

Din punct de vedere funcţional am subîmpărţit această clasă în **10** grupe de instrucţiuni.

Filele grupelor de instrucţiuni ale clasei LOAD—16 se regăsesc în partea a II-a, COMPLEMENTE, § E.4.

5.5. Aventura a 3-a : transferul blocurilor de date

Microprocesorul **Z80** permite, ca folosind o singură instrucțiune, **(LDDR** sau **LDIR)** programatorul să poată disloca blocuri de date de lungime **programabilă** (max. 65536 octeți).

```
7000    LDIR                    ; Im. 68 — Im. 77 (MC)
7002    LD      D,H             ; Im. 78          (MC)
```

Itinerariul nostru nu face altceva decît să urmărească execuția instrucțiunii **LDIR** în modul ciclu mașină. Instrucțiunea **LD D,H** am introdus-o doar ca delimitator.

Condițiile inițiale sînt :

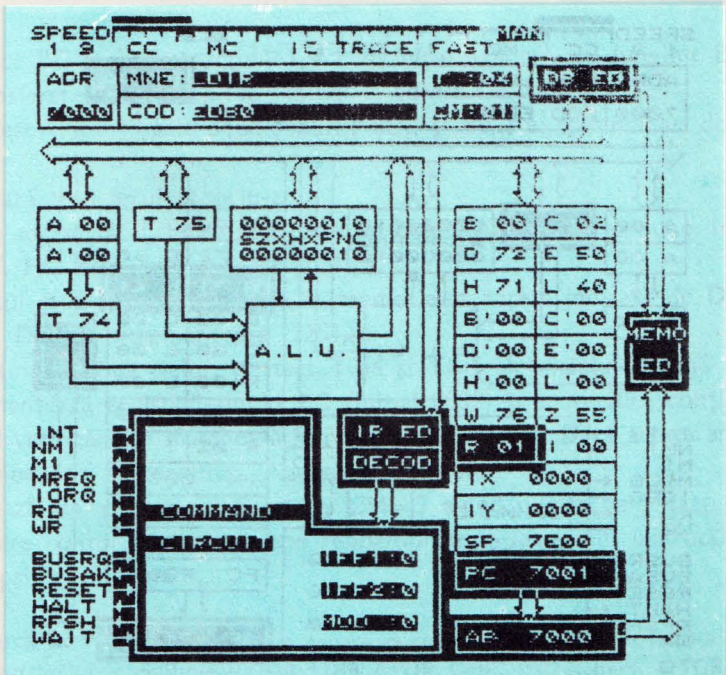
— adresa de început a blocului **sursă** : $HL=7140_H$

— adresa de început a blocului **destinație** : $DE=7250_H$

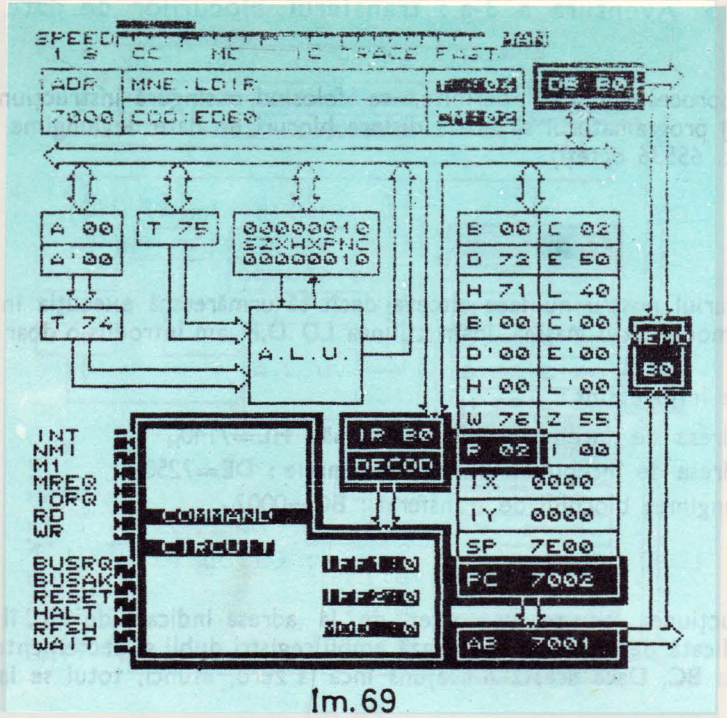
— **lungimea** blocului de transferat : $BC=0002_H$

```
7000    LDIR                    ; Im. 68 — Im. 77 (MC)
```

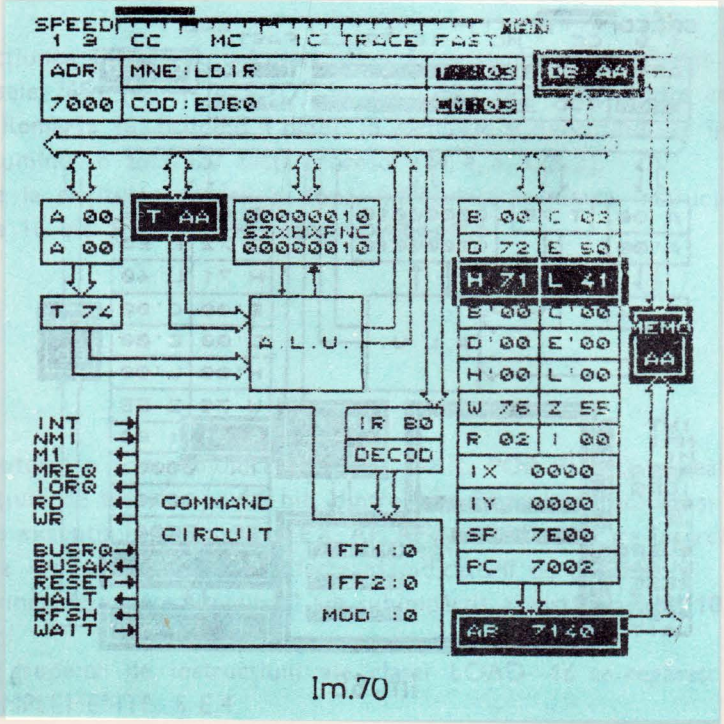
Instrucțiunea citește un octet de la adresa indicată de **HL,** îl înscrie la adresa indicată de **DE,** incrementează ambii regiștri dubli și decrementează registrul dublu **BC.** Dacă acesta n-a ajuns încă la zero, atunci, totul se ia de la în-



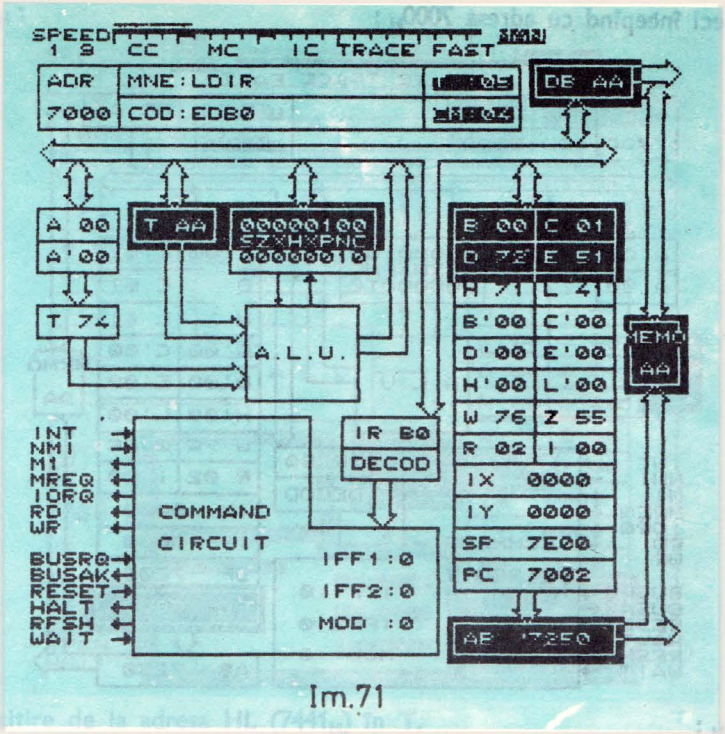
Im. 68


SPEED
1 2
CC
MC
TC
TRACE
FAST
ADR
MNE : LD I R
7000
COD : EDB0
DB B0
A 00
A' 00
T 75
00000010
SZXHXPNC
00000010
T 74
A.L.U.
B 00
C 02
D 72
E 50
H 71
L 40
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 76
Z 55
R 02
I 00
IX 0000
IY 0000
SP 7E00
PC 7002
AB 7001
MEMO
B0
IR B0
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.69


SPEED
1 2
CC
MC
TC
TRACE
FAST
ADR
MNE : LD I R
7000
COD : EDB0
DB AA
A 00
A 00
T AA
00000010
SZXHXPNC
00000010
T 74
A.L.U.
B 00
C 02
D 72
E 50
H 71
L 41
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 76
Z 55
R 02
I 00
IX 0000
IY 0000
SP 7E00
PC 7002
AR 7140
MEMO
AA
IR B0
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.70



Im.71

ceput. Dacă **BC** = 0, instrucțiunea s-a terminat. Blocul de lungime BC—inițial a fost transferat de la adresa **HL**—inițial la adresa **DE**—inițial.

Se citesc mai întîi cele două coduri : ED_H și $B0_H$, executîndu-se două cicluri fetch.

Urmează transferul propriu-zis :

De la adresa **HL** (7140_H) se citește un octet (AA_H) și se depune în registrul tampon **T. HL** este incrementat (7141_H).

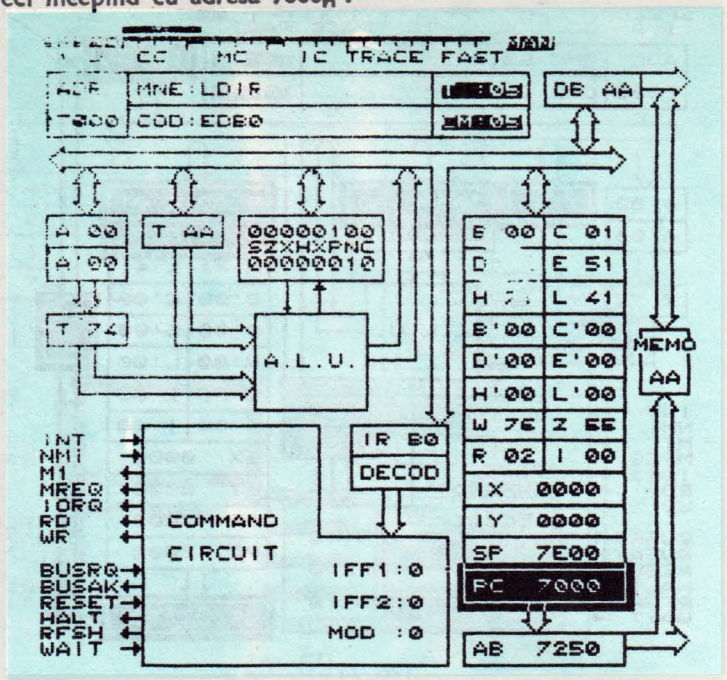
Octetul citit (AA_H) se înscrie în memorie la adresa indicată de **DE** (7250_H) după care **DE** se incrementează (7251_H).

În ultimele două cicluri de tact T=4 și T=5 ale ciclului mașină **(CM=04)** se decrementează registrul dublu **BC** (inițial 0002) și se verifică conținutul lui. Rezultatul verificării se marchează în registrul de flaguri. Dacă acesta ar fi fost 0 instrucțiunea s-ar fi terminat. Așa însă, continuă.

Urmează un ciclu mașină intern **CM=5** avînd durata de 5 tacți procesor, ciclu în care contorul program este repoziționat pe începutul instrucțiunii, care continuă prin reexecutare.

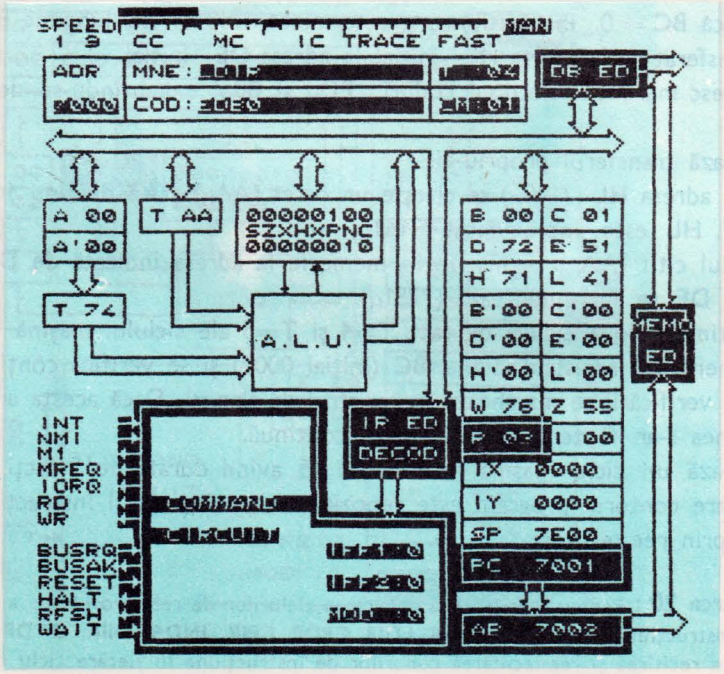
Remarca 20 : Pentru a se asigura generarea ciclurilor de reîmprospătare a memoriilor dinamice, instrucțiunile multiple **(LDDR, LDIR, CPDR, CPIR, INDR, INIR, OTDR, OTIR)** se execută prin recitirea și reexecutarea codurilor de instrucțiune în fiecare ciclu de transfer.

Reluăm deci începînd cu adresa 7000_H :



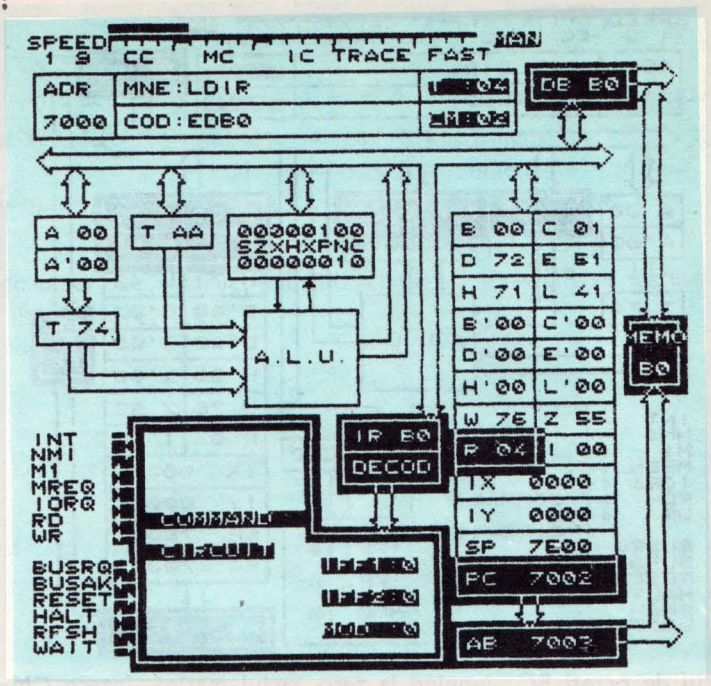
Im.72

Fetch ED_H :



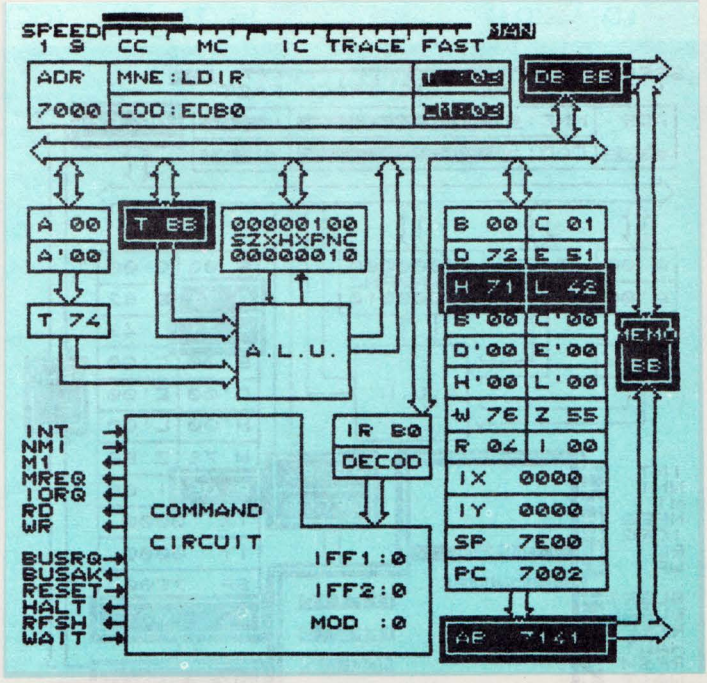
Im.73

Fetch $B0_H$:



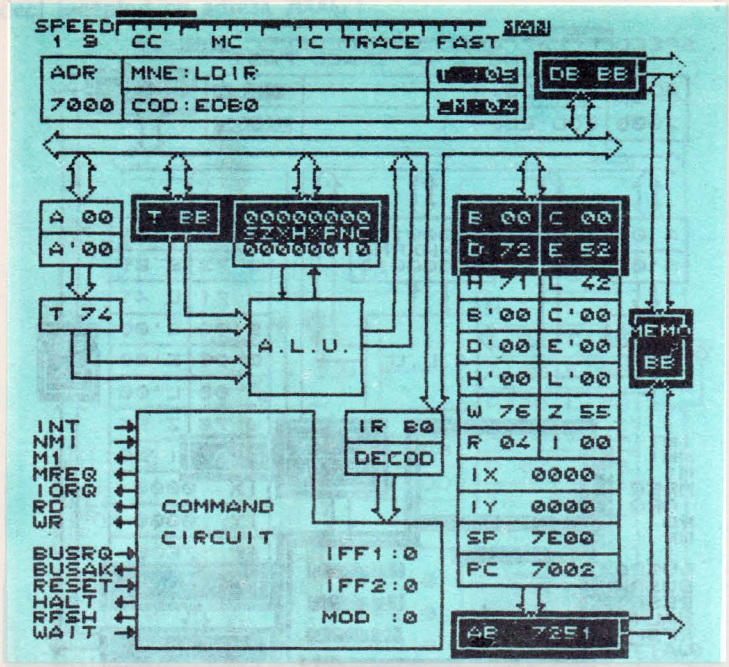
Im. 74

O nouă citire de la adresa HL (7441_H) în T.



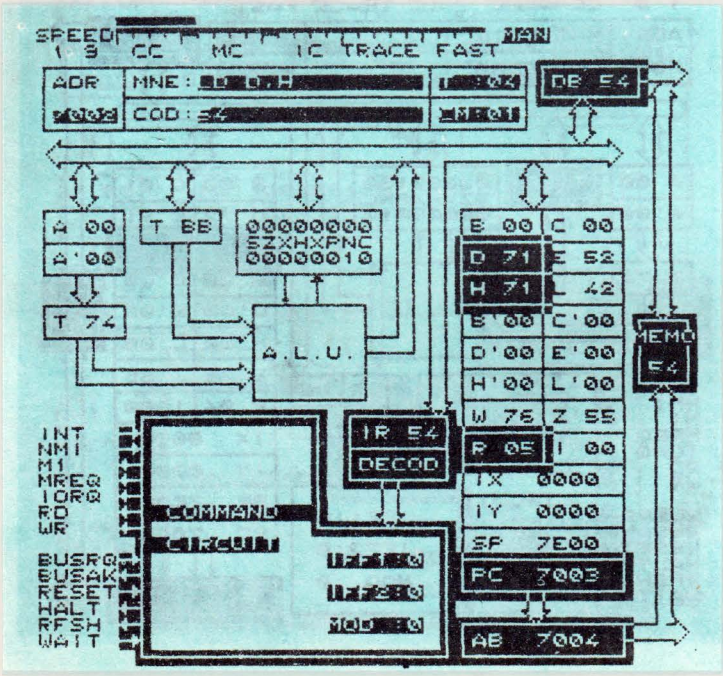
Im. 75

urmată de o nouă scriere în memorie din **T** la adresa **DE** (7551_H)



Im.76

Contorul de octeți **BC** ajungînd la zero, ciclul mașină intern **CM**=05 nu se mai execută, aventura noastră luînd sfîrșit cu execuția următoarei instrucțiuni:



Im.77

5.6. Sinteza clasei LOAD—IDR

Totalul instrucțiunilor din clasă este **4**, împărțită în două grupe :

transfer simplu : **LDD** — load & decrement
LDI — load & increment
transfer multiplu : **LDDR** — load, decrement, repeat
LDIR — load, increment, repeat

Filele grupelor de instrucțiuni din clasa LOAD—IDR se găsesc în partea a II-a, COMPLEMENTE, § E.5.

6

INSTRUCȚIUNI ARITMETICE ȘI LOGICE

Indiferent la ce nu s-ar folosi calculatoarele în zilele noastre, începînd de la crearea și întreținerea bazelor de date, pînă la comanda proceselor industriale, de la asistarea instruirii pînă la asistarea conducerii (**CAX** — cum spunea profesorul Jacques Hebenstreit ; computer aided everything) trăsătura lor comună rămîne aceea că, pe lîngă multe alte facilități, ele știu să și calculeze. Întrebarea, dacă instrucțiunile aritmetice sînt necesare sau nu, nu o mai punem. Instrucțiunile logice apar împreună cu aceste instrucțiuni de nesubstituit, datorită faptului că în aritmetica binară operațiile aritmetice de bază se efectuează tot prin funcții logice. Deci : dacă am să dau la alții, atunci înseamnă că am și eu (nu-i așa ?). Instrucțiunile aritmetice se regăsesc împreună cu cele logice în orice calculator, ele fiind prelucrate de către aceleași circuite electronice : unitatea aritmetică/logică (**ALU**).

Celor care mai contestă importanța software-ului, dacă ei mai există —, le recomandăm ca după studierea atentă a prezentului capitol să încerce să scrie un programaș, folosind instrucțiunile microprocesorului **Z80**, care să calculeze logaritmul natural al oricărui număr real pozitiv, măcar cu o precizie de 6 cifre semnificative. Lucru elementar, pe care-l știe orice calculator personal (deci și cele realizate cu microprocesorul **Z80**), din naștere.

6.1. Aventura a 4-a : aritmetică și logică pe 8 bit

Noi ceilalți, să ne întoarcem la cabana noastră (7000_H) pentru a pregăti o nouă incursiune.

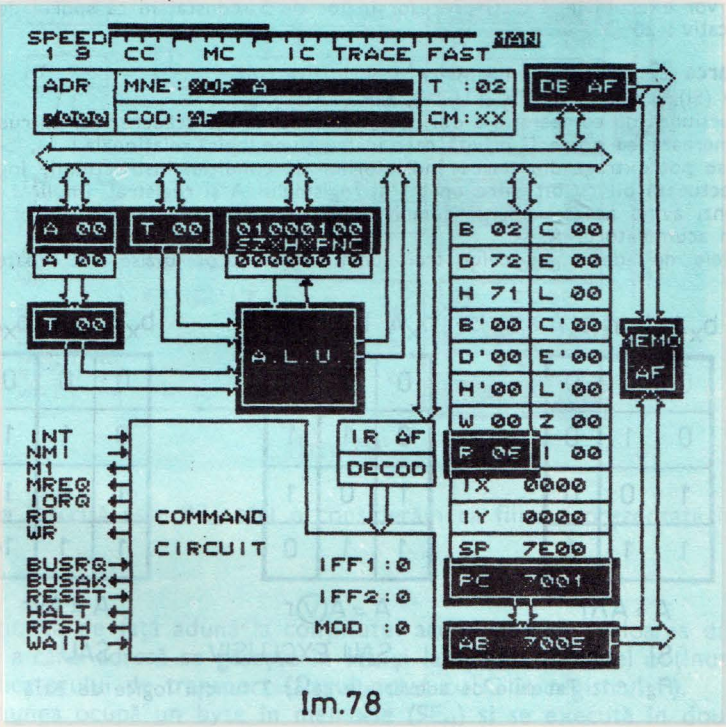
Iată itinerariul ales :

```
7000    XOR   A          ; Im. 78          (IC)
7001    LD    A,(DE)     ; Im. 79 ; 86     (IC)
7002    ADC   A,(HL)     ; Im. 80 ; 87     (IC)
7003    DAA              ; Im. 81 ; 88     (IC)
7004    LD    (HL),A     ; Im. 82 ; 89     (IC)
7005    INC   E          ; Im. 83 ; 90     (IC)
7006    INC   L          ; Im. 84 ; 91     (IC)
7007    DJNZ  F8         ; Im. 85 ; 92     (IC)
7009    JR    NZ,FF      ; Im. 93          (IC)
```

Reamintindu-vă că toate instrucțiunile aritmetice/logice se efectuează între doi parametri, dintre care unul va trebui să fie locat obligatoriu în registrul A, să pornim la drum.

Instrucțiunea realizează operația logică SAU EXCLUSIV între conținutul acumulatorului și el însuși. Rezultatul operației este desigur zero. Aceasta este cea mai scurtă instrucțiune (un octet) pentru ștergerea acumulatorului, și poziționarea adecvată a flagurilor. (LD A,00 ar necesita 2 octeți și nici nu poziționează indicatorii de condiție).

Instrucțiunea XOR r unde r poate fi oricare din regiștri interni, ocupă un byte și durează 4 tacți procesor.



Im.78

Remarca 21. Folosind modul de lucru IC, ne-am fi așteptat ca starea ilustrată în Im. 78 (sfîrșitul instrucțiunii), să fie caracterizată prin CM=1 și T=4. Indicația din Im. 78 (CM=XX), T=2) se cere a fi explicată.

În realitate cei 4 tacți procesor afectați ciclului fetch (**M1**) nu sînt suficienți pentru efectuarea operațiilor aritmetice/logice. Iată de ce :

— în tactul **T3**, codul instrucțiunii se depune în registrul de instrucțiune **IR** și începe decodificarea ;

— în tactul **T4** conținutul registrului selectat și cel al acumulatorului este adus la cele 2 intrări ale **ALU** ;

— pentru a reînscrie rezultatul operației, obținut la ieșirea **ALU**, în acumulator, mai este nevoie de un ciclu de tact suplimentar. În modul cel mai simplu problema s-ar fi putut rezolva suplimentînd ciclul fetch cu încă un tact. În acest caz instrucțiunea s-ar fi terminat în starea CM=1 ; T=5.

Cunoscînd însă faptul că ultima operație de efectuat (transfer **ALU→A**) este o activitate internă procesorului, execuția ei poate fi amînată pe durata ciclului mașină următor.

ciclul mașină care este primul ciclu mașină al unei instrucțiuni următoare. De aceea notația CM=XX. Dar ce înseamnă T=2 ? Aceasta este o moștenire de la microprocesorul **I8080,** la care transferul **ALU→A** nu s-a putut efectua în primul ciclu mașină T=1 al instrucțiunii următoare, datorită faptului că magistrala internă de date a procesorului era ocupată cu emiterea „cuvîntului de stare". Deci transferul **ALU→A** a trebuit să fie amînat pe cel de-al doilea ciclu de tact, T=2. În documentațiile publicate de către fabricanții microprocesorului **Z80** nu se face nici o referire la acest detaliu. De aceea, în **VISIBLE—Z80** noi am adoptat varianta publicată, cea de la **Intel.** Menționăm că după părerea noastră în microprocesorul **Z80** transferul **ALU→A** se efectuează în primul ciclu de tact (T=1) al noii instrucțiuni, căci spre deosebire de **I8080,** magistrala internă de date a microprocesorului **Z80** este liberă în acel moment.

Tehnica folosită, cea de a amîna finalizarea unei instrucțiuni (sau a unei operații interne), pe durata ciclului mașină următor, atunci cînd procesorul execută manevre externe, se numește *suprapunere de cicluri mașină* (sau *„furt de ciclu"*). Scopul suprapunerii de cicluri mașină este cel de a spori viteza de execuție. Știind că astfel instrucțiunile aritmetice/logice se vor executa în 4 tacți procesor în loc de 5, constatăm că sporul de viteză este semnificativ : 20%.

Remarca 22. Instrucțiunile logice propriu-zise sînt :
AND (SI), **XOR** (SAU EXCLUSIV), **OR** (SAU)

Instrucțiunea de comparație a doi octeți este de fapt o scădere, dar întrucît rezultatul ei nu se generează, ea poate fi privită ca o instrucțiune logică relațională : =, >, <. Aceste informații se pot extrage din starea indicatorilor de condiție. Instrucțiunile logice propriu-zise, se efectuează bit cu bit între un bit al registrului A și registrul implicat în operație, biții operanzi avînd aceeași poziție semnificativă în cadrul octetului. Octetul rezultant se înscrie în acumulatorul A.

Tabelele de adevăr ale celor trei funcții logice propriu-zise sînt redate în fig. 6.1.

b_xA	b_xr	b_xA
0	0	0
0	1	0
1	0	0
1	1	1

$A = A \wedge r$

SI

b_xA	b_xr	b_xA
0	0	0
0	1	1
1	0	1
1	1	0

$A = A \textcircled{v} r$

SAU EXCLUSIV

b_xA	b_xr	b_xA
0	0	0
0	1	1
1	0	1
1	1	1

$A = A \vee r$

SAU

Fig. 6.1. Tabelele de adevăr ale celor 3 funcții logice de bază

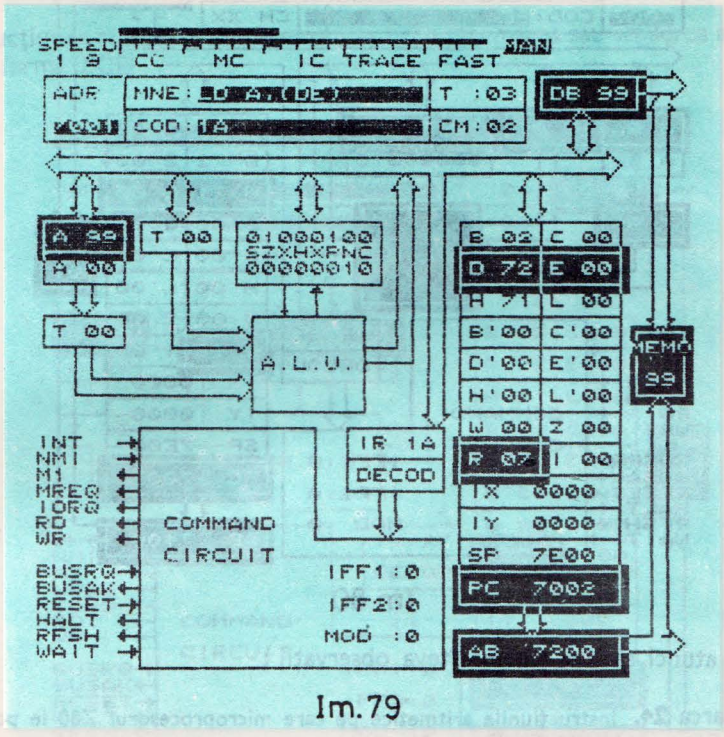
Remarca 23. În afara destinației lor de bază, cea de a calcula expresii logice, funcțiile logice pot fi folosite eficient pentru a șterge, a înscrie sau pentru a completa un bit sau un grup de biți ai unui octet.

Fie octetul pe care dorim sa-l manevrăm, în registrul **A.** Notăm biții interesanți cu b_i. Într-unul din regiștri interni depunem un octet numit mască, în care biții b_i vor avea o stare, iar toți ceilalți starea complementară. Iată operația și tipul de mască recomandat :

Tabela c.t. **Tehnici de mascare a biților**

Efectul dorit	Operația folosită	Starea biților de mască b_i
Ștergerea unor biți b_i	**AND** (SI)	$b_i = 0$
Înscrierea unor biți b_i	**OR** (SAU)	$b_i = 1$
Inversarea unor biți b_i	**XOR** (SAU EXCLUSIV)	$b_i = 1$

Aruncînd o ultimă privire fugitivă asupra Im.78 observăm că flagul **Z (zero)** s-a înscris. Încărcăm în acumulator un număr din memorie, așa cum se vede în Im.79 folosind instrucțiunea LD A,(DE).



Im.79

Valoarea înscrisă este 99_D căci o considerăm ca fiind reprezentată în BCD.

Instrucțiunea de față adună la conținutul acumulatorului valoarea din celula de memorie a cărei adresă se găsește în **HL** și la rezultatul astfel obținut adaugă valoarea indicatorului de transport (flagul notat cu **C** în registrul **F**).

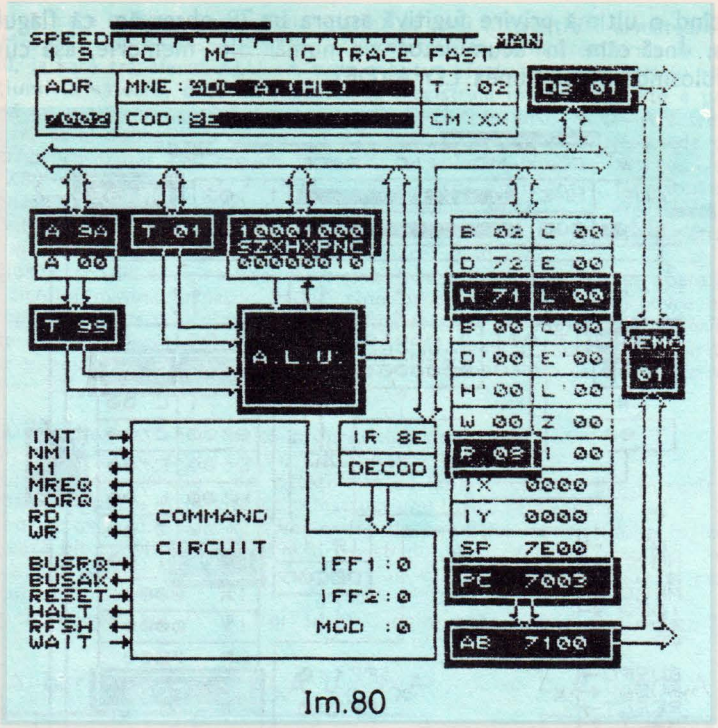
Instrucțiunea ocupă un byte în memorie ($9E_H$) și se execută în două cicluri mașină însumînd 7 tacți (4, 3).

Întrucît în Im. 80 instrucțiunea a fost executată în modul IC, ciclul mașină indicat este, ca și la instrucțiunea de mai sus **(XOR A)**, ciclul ascuns (CM = = XX, T=2).

Octetul citit din memorie se află în registrul tampon din dreapta, vechiul conținut al acumulatorului în registrul tampon din stînga, iar rezultatul în **A**.

Într-adevăr $A = A + (HL) + Cy = 99 + 01 + 0 = 9A_H$

Remarcăm faptul că **ALU** nu știe dorința noastră și anume că 99 este un număr BCD. El tratează toate numerele ca și binare. De aceea pentru a obține rezultatul sub forma dorită (BCD), după efectuarea operației aritmetice se va aplica o corecție prin instrucțiunea **DAA** așa cum vom vedea la adresa 7003_H.



Im.80

Pînă atunci să mai facem cîteva observaţii :

Remarca 24. Instrucţiunile aritmetice pe care microprocesorul Z80 le poate executa sînt :

ADD — adunare simplă
ADC — adunare cu adunarea transportului
SUB — scădere simplă
SBC — scădere cu scăderea transportului

Al doilea operand poate fi locat fie într-un registru intern, fie în memorie. În funcţie de tipul de adresare folosit, distingem 3 grupe de instrucţiuni :

a) operaţii aritmetice cu adresare *implicită* (op_2 în registru)

ADD	**A, r**	A = A + r
ADC	**A, r**	A = A + r + Cy
SUB	**r**	A = A − r
SBC	**A, r**	A = A − r − Cy

unde **r** poate fi unul din regiştrii **B, C, D, E, H, L** sau **A,** iar **Cy** este indicatorul de transport.

b) operaţii aritmetice cu adresare *indirectă* sau *indexată* (op_2 în memorie)

ADD	**A, (mem)**	A = A + (mem)
ADC	**A, (mem)**	A = A + (mem) + Cy
SUB	**(mem)**	A = A − (mem)
SBC	**A, (mem)**	A = A − (mem) − Cy

unde **mem** poate fi **HL, IX+IND** sau **IY+IND.**
(Regiştri BC şi DE nu pot fi folosiţi ca mijloc de adresare indirectă, la această clasă de instrucţiuni !)

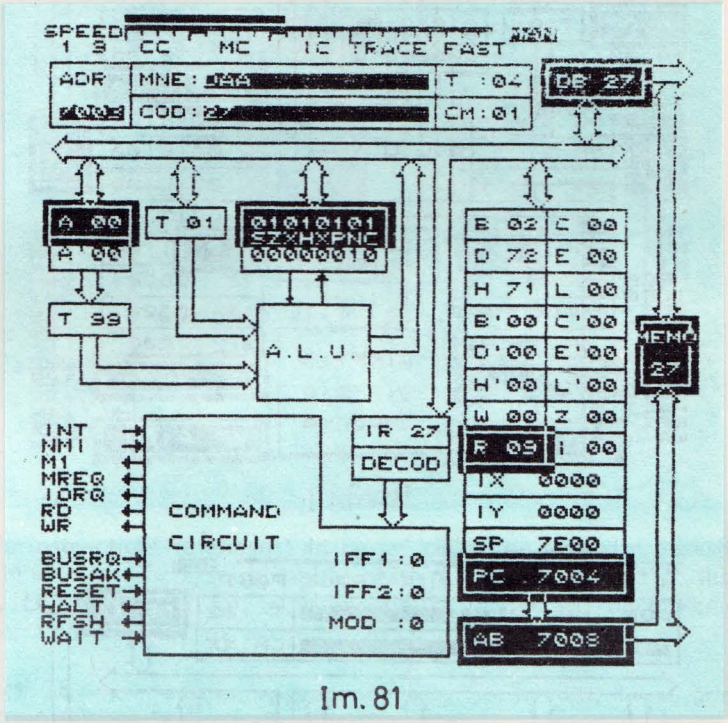
c) operaţii aritmetice cu adresare *imediată* (op_2 în memorie, în cîmpul instrucţiunii)

ADD	**A, n**	A = A+n
ADC	**A, n**	A = A+n+Cy
SUB	**n**	A = A − n
SBC	**A, n**	A = A − n − Cy

unde **n** este un număr binar de 8 bit [0,255].

Suindu-ne din nou pe cal...

Instrucţiunea ajustează numărul obţinut din însumarea sau scăderea a 2 numere BCD, la formă BCD



Im. 81

La adunarea precedentă, ne-am aşteptat ca rezultatul să fie : $99_D + 01 = 100_D$, dar s-a obţinut $9A_H$. După instrucţiunea **DAA** (Im. 81) rezultatul din acumulator a fost reajustat la valoarea BCD corectă (**A** = 00 ; **Cy**=1 ⇒ rez = = 100). De aceea ea se numeşte **DAA** (Decimal Adjust Accumulator).

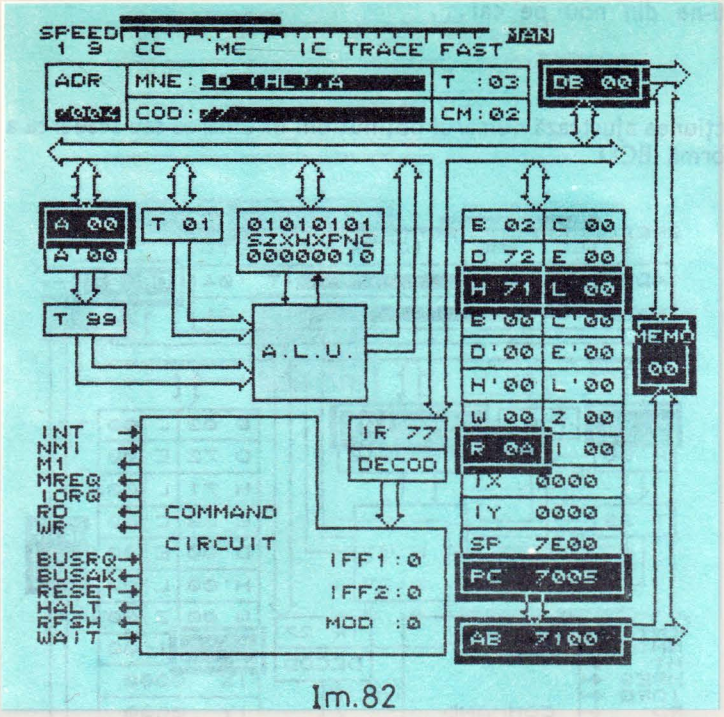
Folosind flagurile **N**, **H** şi **Cy**, reajustarea se face aproape instantaneu, instrucţiunea executîndu-se într-un singur ciclu maşină (4 taçţi procesor).

Remarca 25 : Dacă numerele implicate într-o operaţie aritmetică nu sînt ambele numere BCD, atunci nici **DAA** nu poate ajusta corect rezultatul.

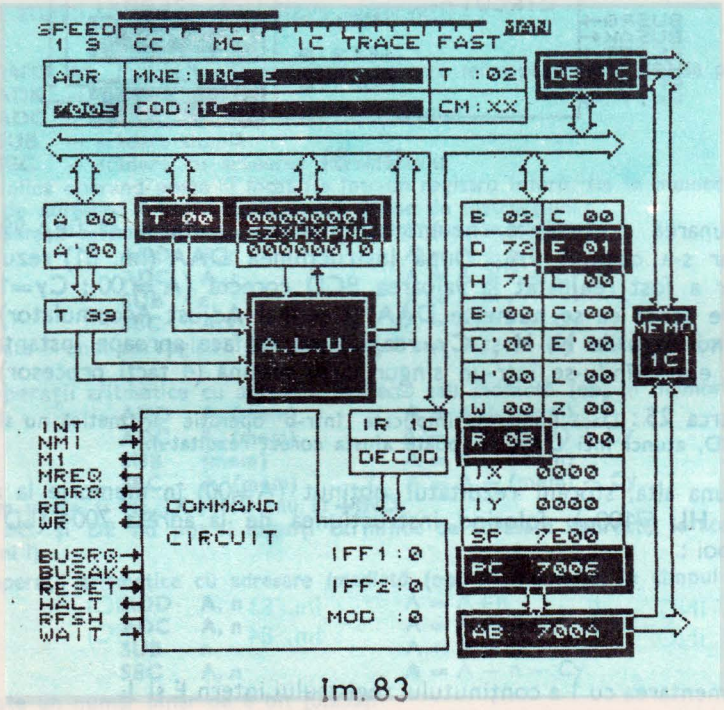
Pînă una alta, stocăm rezultatul obţinut (A=00) în memorie la adresa indicată de **HL** (7100_H), folosind instrucţiunea de la adresa 7004 (LD (HL), A). Iar apoi :

```
7005    INC    E        ; Im. 83    (IC)
7006    INC    L        ; Im. 84    (IC)
```

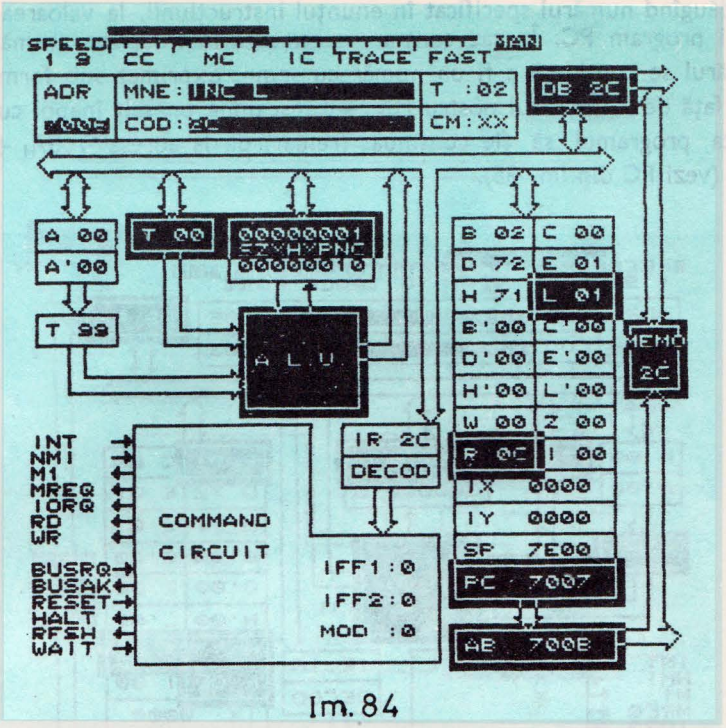
Incrementarea cu 1 a conţinutului registrului intern **E** şi **L**


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE: LD (HL),A
T :03
DB 00
COD:
CM:02
A 00
A'00
T 01
01010101
SZXHXPNC
00000010
B 02
C 00
D 72
E 00
H 71
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 0A
I 00
IX 0000
IY 0000
SP 7E00
PC 7005
AB 7100
T 99
A.L.U.
MEMO
00
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IR 77
DECOD
IFF1:0
IFF2:0
MOD :0

Im.82


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE: INC E
T :02
DB 1C
COD:
CM:XX
A 00
A'00
T 00
00000001
SZXHXPNC
00000010
B 02
C 00
D 72
E 01
H 71
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 0B
I 00
IX 0000
IY 0000
SP 7E00
PC 7006
AB 700A
T 99
A.L.U.
MEMO
1C
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IR 1C
DECOD
IFF1:0
IFF2:0
MOD :0

Im.83



Im. 84

Ambele instrucțiuni se execută în cîte 4 cicluri de tact, dar necesită fizic 5. Ultimul (CM = ××, T=2) se suprapune peste primul ciclu mașină al instrucțiunii următoare. Din Im. 83 și Im. 84 observăm că în operația de incrementare se implică doar una din căile de acces ale ALU.

Remarca 26 : Incrementarea regiștrilor simpli se face în ALU, dar nu prin adunarea cu 1 ci printr-un automat hardware specializat.

Aceeași observație este valabilă și pentru instrucțiunile de decrementare DEC a căror structură și execuție este similară cu cea a instrucțiunilor INC.

Forma generală este : INC r
DEC r

unde r poate fi B, C, D, E, H, L sau A.

Remarca 27 : Folosind adresarea indirectă sau cea indexată, este posibilă incrementarea sau decrementarea unor celule de memorie, dar numai prin utilizarea mereu acelorași regiștri privilegiați HL, IX, IY.

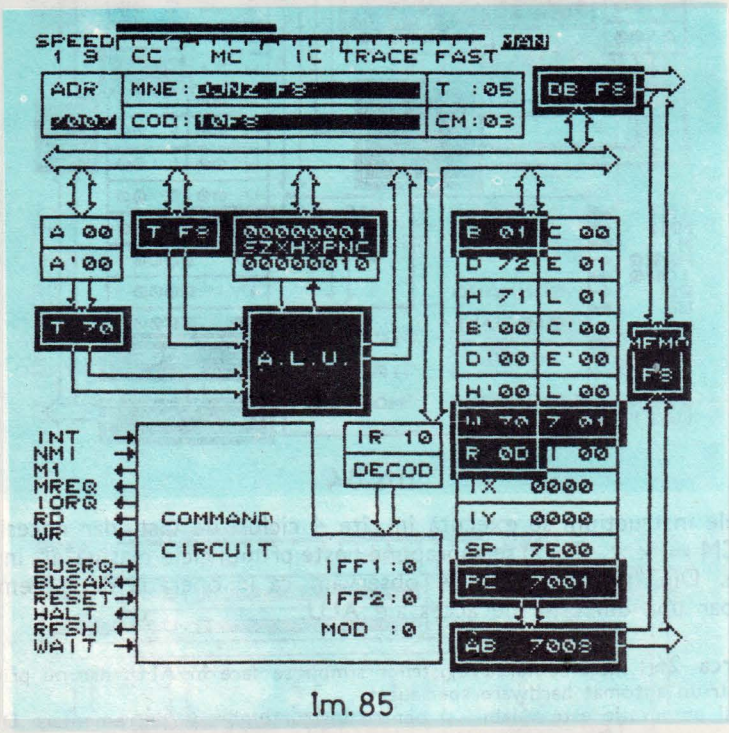
INC HL	respectiv	DEC HL
INC (IX+IND)		DEC (IX+IND)
INC (IY+IND)		DEC (IY+IND)

Închidem aici secvența de aritmetică într-o buclă, folosind o instrucțiune care face apel atît la resursele instrucțiunilor aritmetice cît și a celor de salt.

Instrucțiunea de față (Decrement Jump if Not Zero), decrementează conținutul registrului B. Dacă rezultatul este diferit de zero se execută un salt

relativ, adăugînd numărul specificat în enunțul instrucțiunii, la valoarea curentă a contorului program PC. În caz contrar se execută instrucțiunea următoare.

Numărul se consideră a fi un număr cu semn, exprimat sub forma complementului față de 2. În cazul nostru F8=—8 înseamnă un salt înapoi cu 8 adrese, urmînd ca programul să fie continuat (reluat) de la adresa $7007_H + 2 - 8 = 7001_H$. (vezi PC din Im. 85).

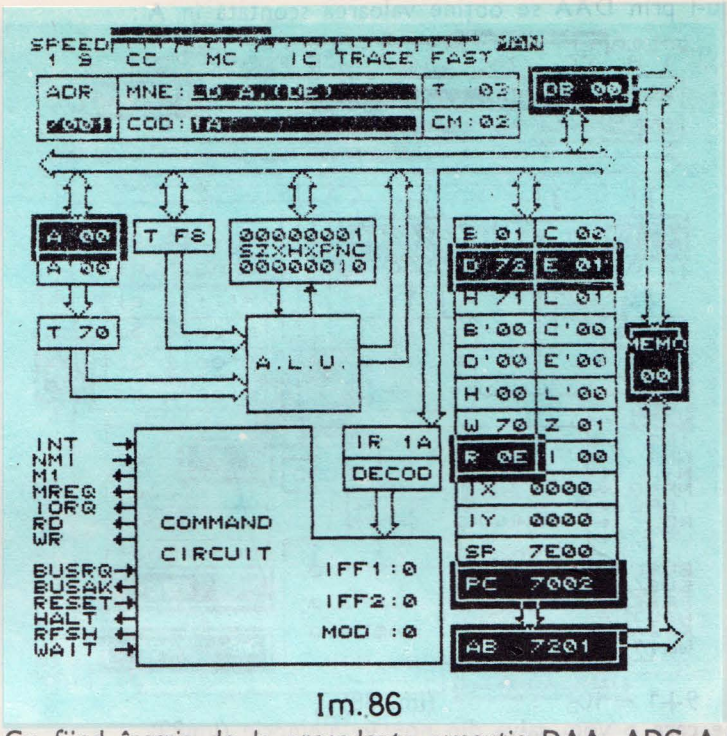


Im. 85

Oprindu-ne în modul IC la sfîrșitul execuției acestei instrucțiuni constatăm că DJNZ ocupă în memorie 2 octeți și se execută în trei cicluri mașină, însumînd 13 (5,3,5) tacți procesor. Ultimul ciclu mașină (M3) este un ciclu intern și se folosește la calculul sumei PC + depl. Dacă B = 0 saltul nu se mai efectuează și deci nici ultimul ciclu mașină (M3) nu mai este necesar, durata de execuție a instrucțiunii reducîndu-se la 8 cicluri de tact (5,3).

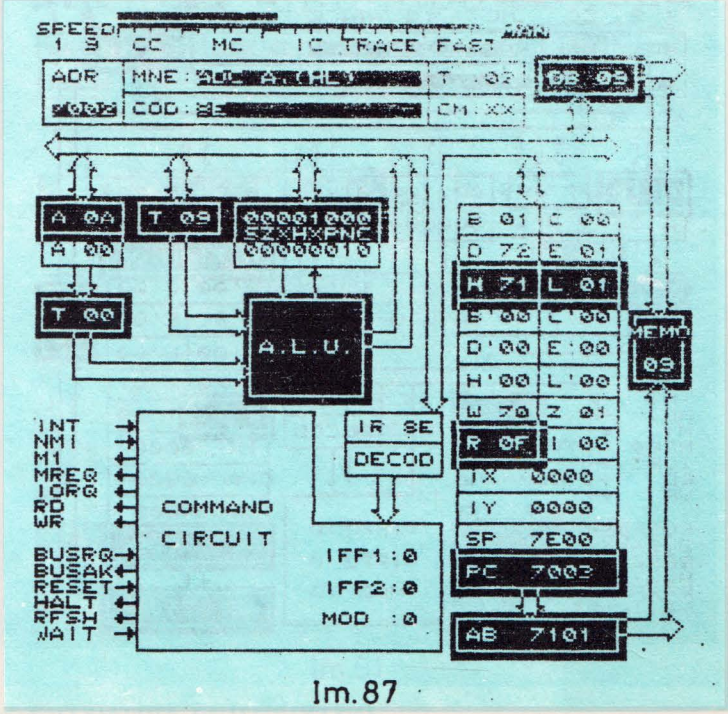
Remarca 28: DJNZ este o instrucțiune unicat, ea decrementînd totdeauna registrul **B.** Datorită faptului că deplasamentul se specifică pe un singur octet ca *număr cu semn,* ecartul saltului este limitat la [+127, —128] față de adresa instrucțiunii imediat următoare. (În momentul calculului adresei de salt, **PC** conține deja adresa instrucțiunii următoare. Toate salturile relative ale microprocesorului **Z80** (le vom cunoaște) sînt limitate la acest interval.

Cum în aventura noastră B=1, diferit de zero, reluăm execuția instrucțiunilor începînd cu adresa 7001_H.



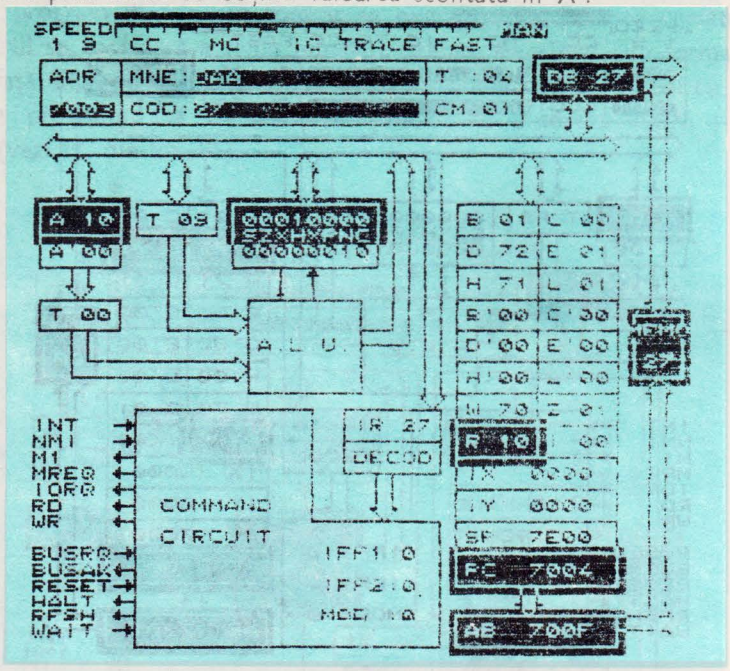
Im. 86

Flagul Cy fiind înscris de la precedenta execuție DAA, ADC A,(HL) din Im. 87 va genera rezultatul : $0+9+1=0A_H$.



Im. 87

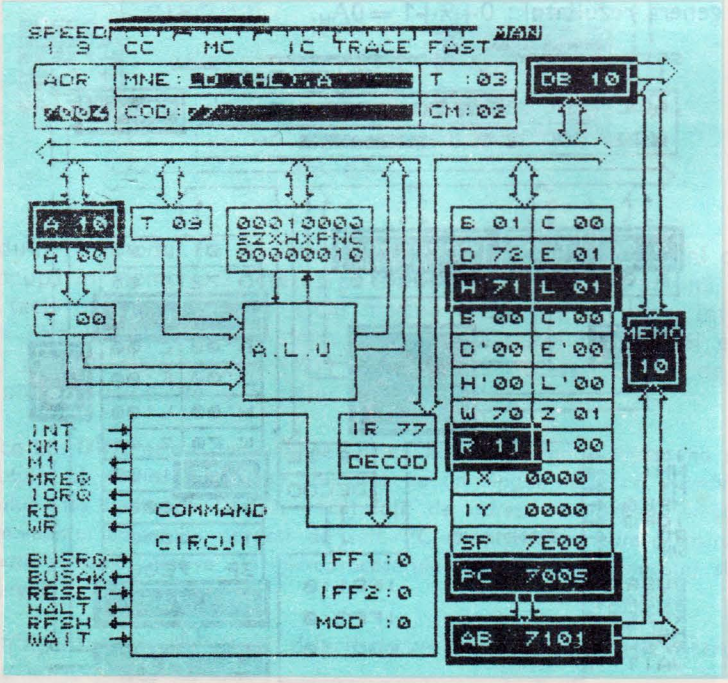
Reajustîndu-l prin DAA se obține valoarea scontată în A :



Im. 88

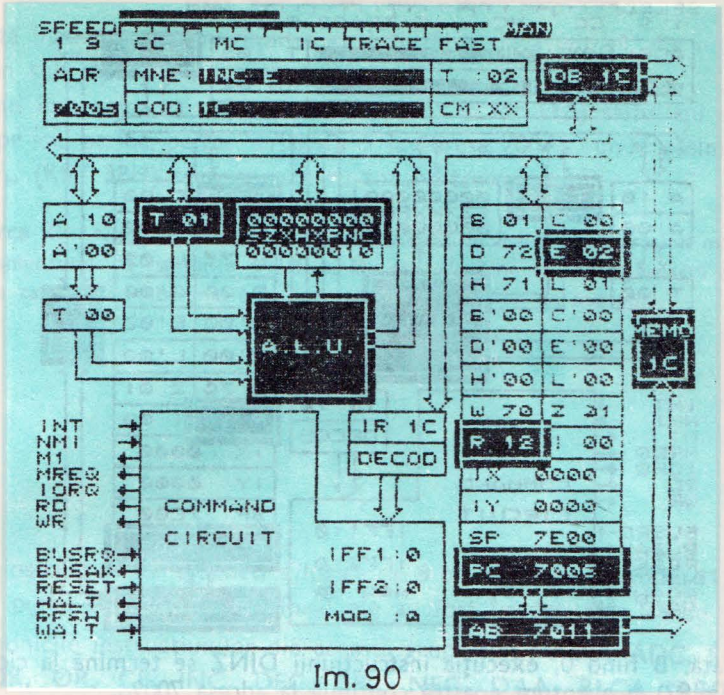
$9+1 = 10_D$ (Im. 88)

Valoare pe care o vom salva din nou în memorie (Im.89).



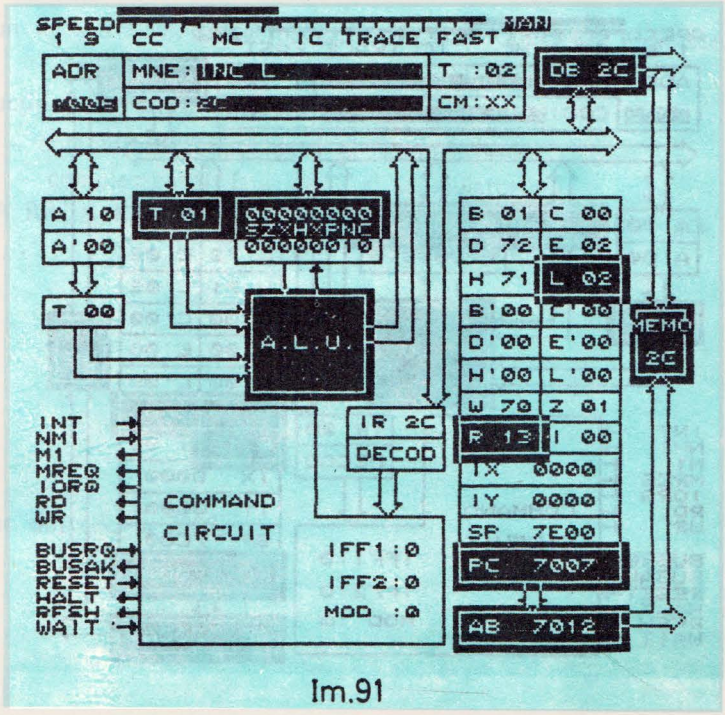
Im. 89

După incrementarea registrilor E



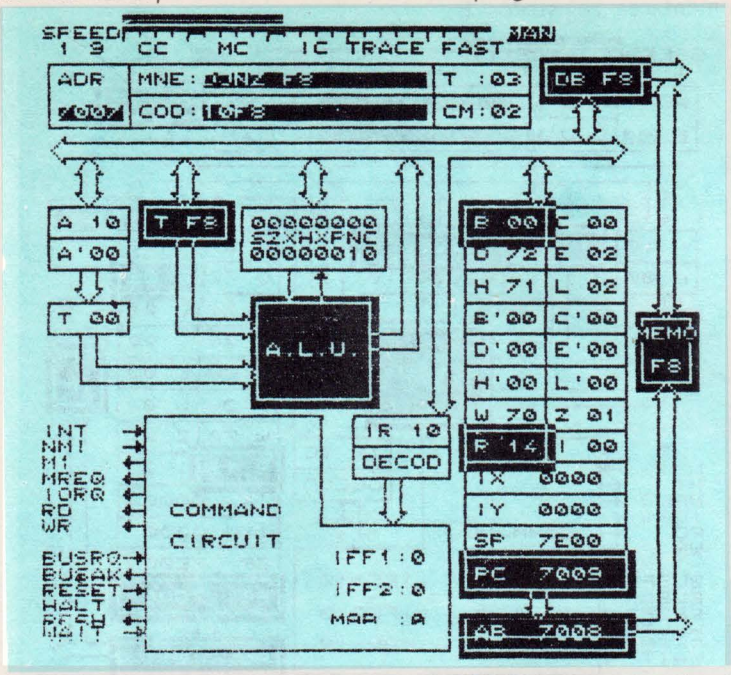
Im. 90

și L,



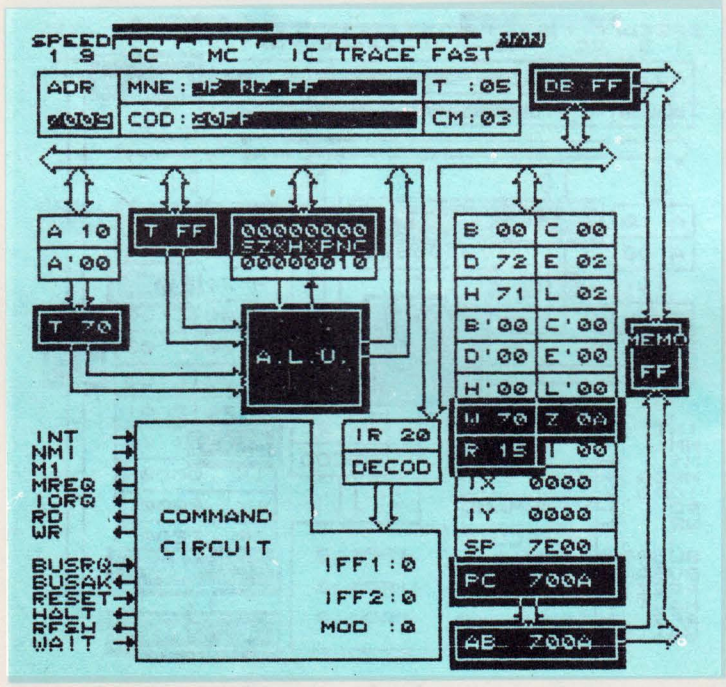
Im.91

testăm din nou condiția de terminare a ciclului program.



Im.92

De astă dată, B fiind 0, execuția instrucțiunii **DJNZ** se termină la ciclul mașină **M2** (vezi Im.92) și aventura noastră continuă la adresa 7009_H.



Im.93

Această instrucțiune am gîndit-o ca închizător de aventură, dorind ca ea să execute un salt „pe loc", la infinit.

Greșind însă valoarea deplasamentului (FF_H în loc FE_H) saltul nu se efectuează „tot pe loc, pe loc, pe loc" (7009_H), ci la adresa $700A_H$, unde găsim „instrucțiunea" FF_H **(RST 38H)** ceea ce este cu totul altceva !

Remarca 29 : Ne-a fost dat să cunoaștem softiști de înaltă clasă greșind mereu valoarea deplasamentului la salturile relative. De aceea e bine să cedăm această operație de rutină unuia care se pricepe mai bine, programului asamblor de exemplu.

6.2. Sinteza clasei AR/LOG—8

Clasa instrucțiunilor aritmetice/logice pe 8 bit cuprinde în total **115** instrucțiuni. Din punct de vedere funcțional ele au fost împărțite în **26** de grupe.

Mnemonicile instrucțiunilor din această grupă sînt : **ADD, ADC, SUB, SBC, AND, XOR, OR, CP, INC, DEC, CPL, NEG, DAA, RLCA, RRCA, RLA, RRA.**

Într-un fel sau altul, toate instrucțiunile clasei afectează indicatorii de condiție.

Instrucțiunile neatinse în aventura a 4-a sînt :

CPL — complementul față de 1 al acumulatorului
NEG — complementul față de 2 al acumulatorului
RLCA, RRCA, RLA, RRA — rotiri la stînga și la dreapta, „cu" respectiv „prin" carry a acumulatorului.

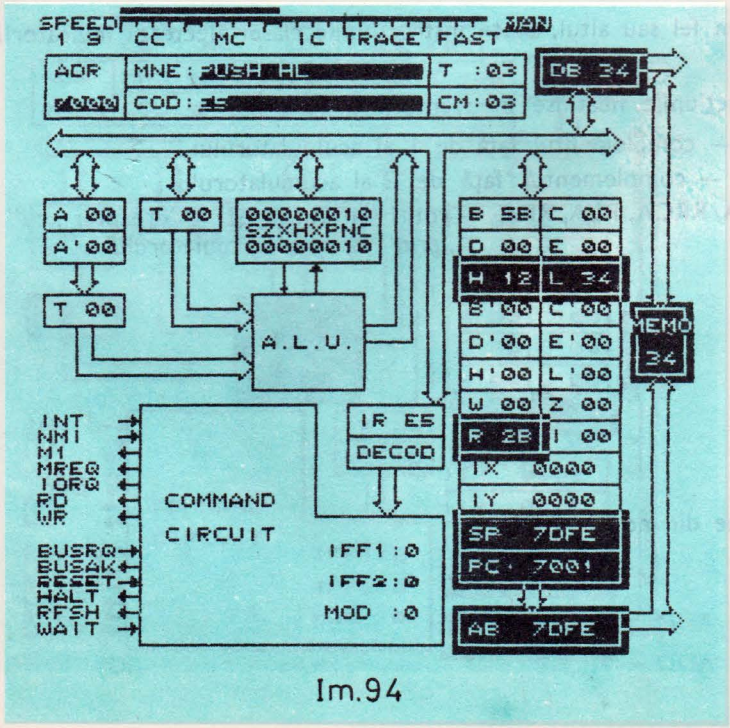
6.3. Aventura a 5-a : aritmetică pe 16 bit

Iată-ne din nou la 7000_H.

```
7000    PUSH    HL          ; Im.94     (IC)
7001    ADD     HL,HL       ; Im.95     (IC)
7002    ADD     HL,HL       ; Im.96     (IC)
```

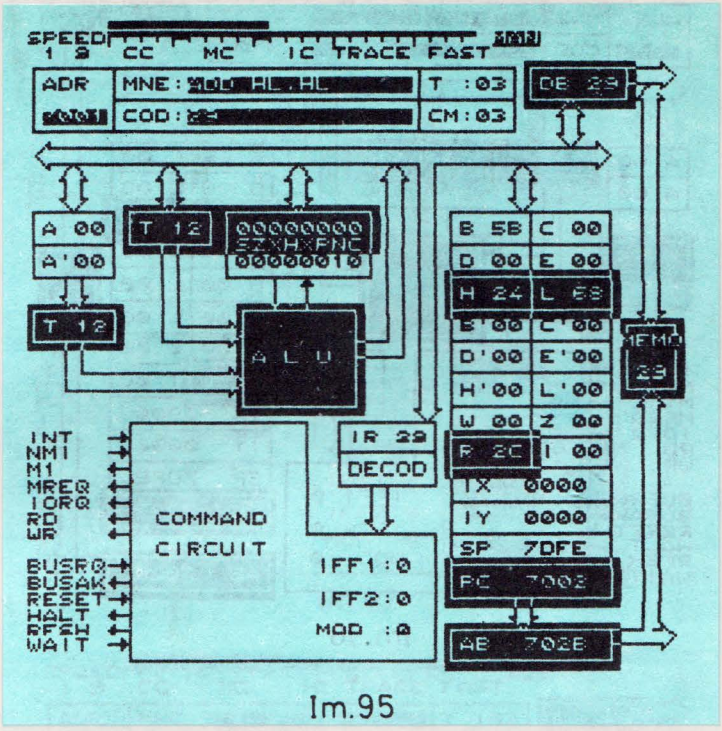
Secvența aceasta este o subrutină, fiindcă se termină cu instrucțiunea **RET**. Aventura de față s-ar putea să-i plictisească pe cititorii care ne-au urmărit atent pînă la acest punct, fiindcă baleind programașul de mai sus, li se pare totul clar. Să-l parcurgem totuși cu ajutorul lui **VISIBLE—Z80,** știind că chiar și o plimbare în propria-ți curte poate fi cîteodată reconfortantă.

Salvăm conținutul registrului dublu **HL** pe stivă,



Im.94

și apoi îl adunăm la el însuși :



Im.95

Oprindu-ne la sfîrșitul celei de-a doua instrucțiuni, **ADD HL,HL** constatăm că ea se codifică pe un byte, iar execuția durează **3** cicluri mașină însumînd **11** tacți procesor **(4,4,3)**. **M2** și **M3** sînt cicluri interne menite să efectueze adunările și transferurile aferente.

Din faptul că registrul **A** n-a fost implicat pe parcursul instrucțiunii (căsuța lui nu s-a înegrit în Im.95) deducem o concluzie importantă.

Remarca 30 : În aritmetica pe 16 biți a microprocesorului **Z80**, rolul aculumulatorului este preluat de registrul dublu **HL** și confrații săi de breaslă **IX** și **IY**.

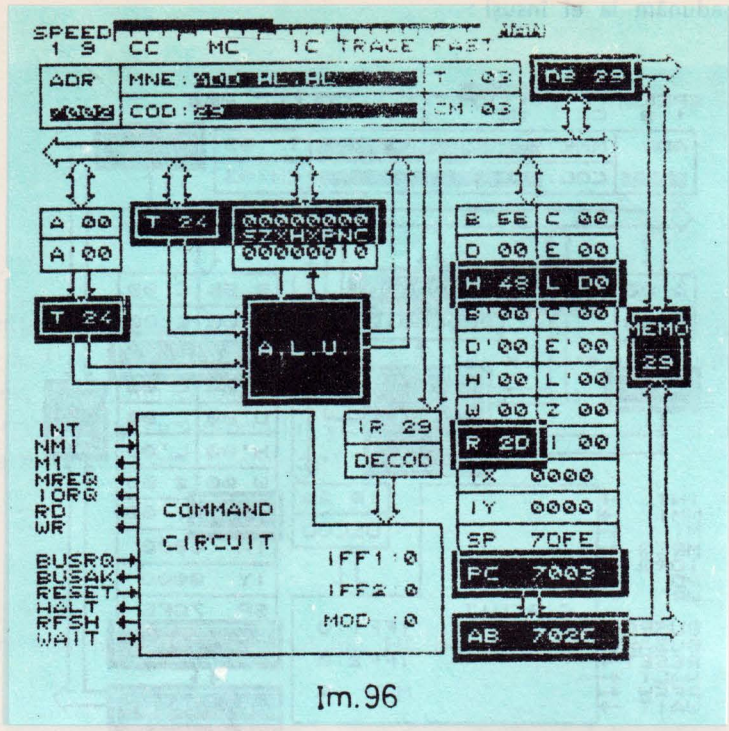
Știind că în aritmetica binară înmulțirea cu 2 este echivalentă cu o deplasare la stînga, corelată cu inserarea unui bit 0 pe poziția cea mai puțin semnificativă, constatăm că **ADD HL, HL** poate fi privit ca o deplasare pe 16 bjt cu o poziție la stînga :

1234_H	—	0001	0010	0011	0100
2468_H	—	0010	0100	0110	1000

În aritmetica pe 8 bit, aceeași funcție s-ar fi realizat prin **ADD A, A.**

7002 ADD HL,HL ; Im.96 (IC)

Repetăm adunarea **ADD HL,HL** pentru a vă sugera ideea că a repeta de n ori această instrucțiune are același efect ca și HL $*$ 2^n



Im.96

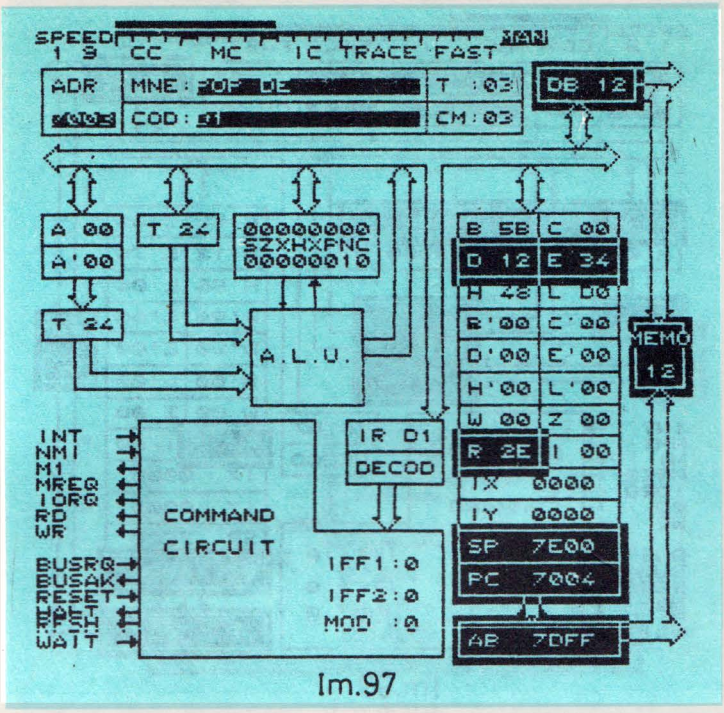
Citim în registrul **DE** ultima valoare depusă pe stivă (7000 **PUSH HL**). Iar apoi efectuăm o adunare clasică a registrului **DE** cu **HL**, obținînd rezultatul în **HL**.

Remarca 31 : Aritmetica pe 16 biți a microprocesorului **Z80** admite un singur tip de adresare, cel implicit. Al doilea operand va fi plasat într-unul din regiștrii **BC, DE, HL** sau **SP**.

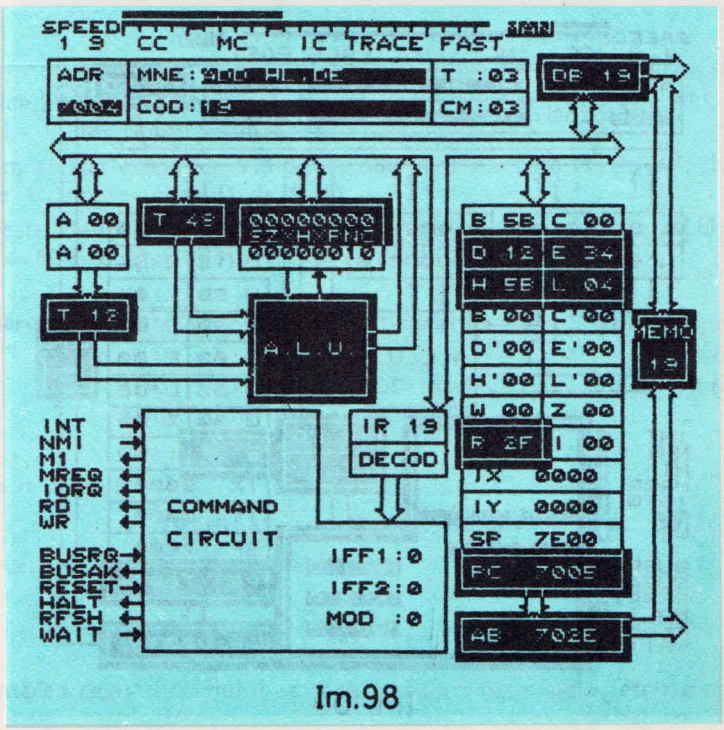
Remarca 32 : Instrucțiunea **ADD HL, SP** precedată de o ștergere a registrului **HL** este una din puținele instrucțiuni, prin care se poate determina, în cursul execuției unui program, valoarea curentă a indicatorului de stivă **SP**. La microprocesorul **I8080**, instrucțiunea echivalentă **DAD SP** era unică).

Remarca 33 : În rutinele de aritmetică este necesară adesea ștergerea indicatorului de transport **Cy**. Cum nu există o instrucțiune dedicată acestui scop, putem folosi instrucțiunea **OR A** sau **AND A** cu același efect.

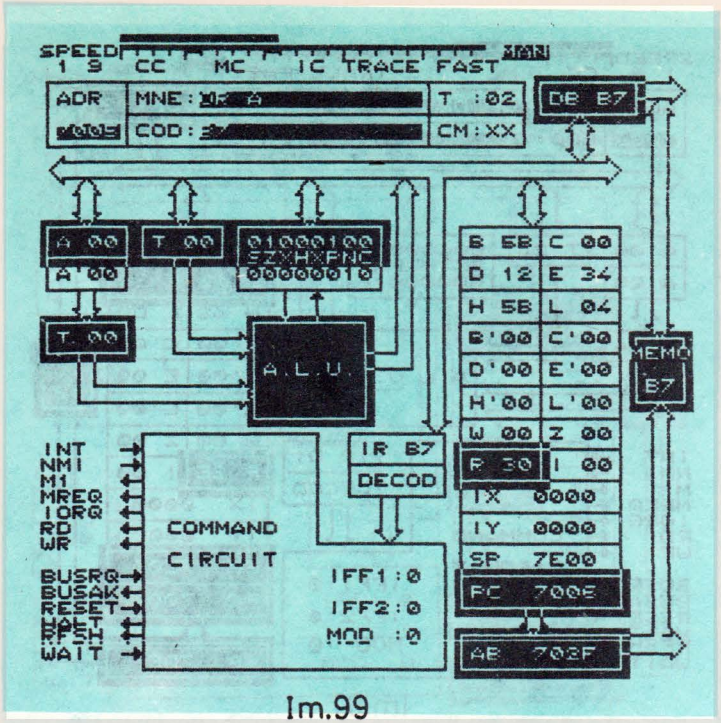
Ajungînd la imaginea nr. 100, să-i dăm cezarului ceea ce i se cuvine, derulînd din nou o instrucțiune în modul cïclu mașină **(MC)**.


SPEED
1 9 CC MC IC TRACE FAST
MAN
ADR
MNE:
COD:
T :03
CM:03
DB 12
A 00
A'00
T 24
00000000
SZXHXPNC
00000010
T 24
A.L.U.
B 5B
C 00
D 12
E 34
H 48
L D0
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 2E
I 00
IX 0000
IY 0000
SP 7E00
PC 7004
AB 7DFF
MEMO
12
IR D1
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

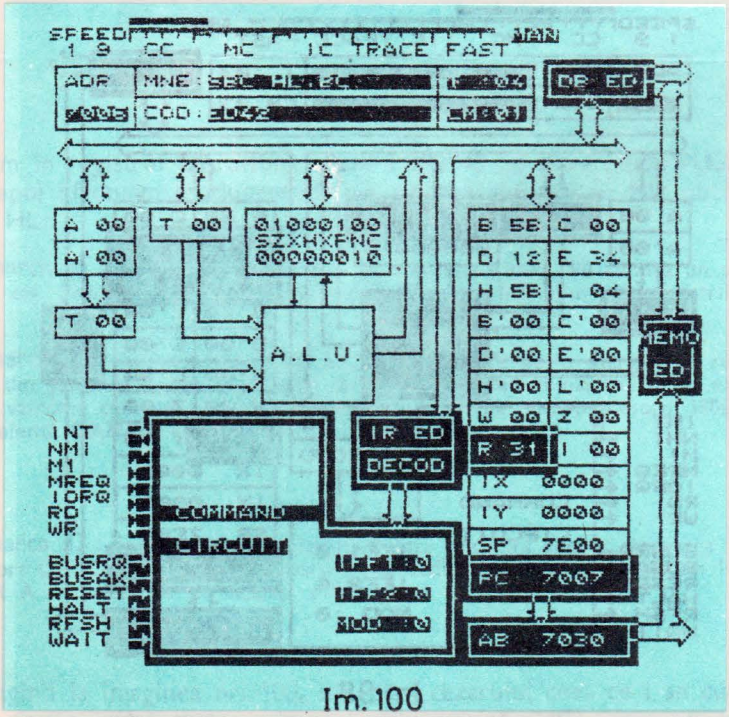
Im.97


SPEED
1 9 CC MC IC TRACE FAST
MAN
ADR
MNE:
COD:
T :03
CM:03
DB 19
A 00
A'00
T 48
00000000
SZXHXPNC
00000010
T 12
A.L.U.
B 5B
C 00
D 12
E 34
H 5B
L 04
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 2F
I 00
IX 0000
IY 0000
SP 7E00
PC 7005
AB 702E
MEMO
19
IR 19
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

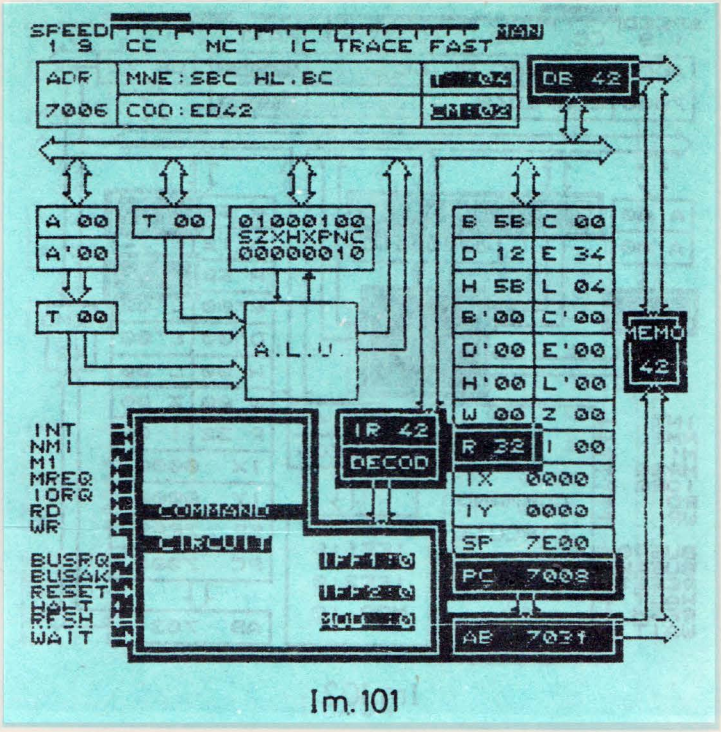
Im.98


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
MNE :
T :02
DB B7
COD :
CM :XX
A 00
A' 00
T 00
01000100
SZXHXPNC
00000010
T 00
A.L.U.
B 5B
C 00
D 12
E 34
H 5B
L 04
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 00
Z 00
R 30
I 00
IX 0000
IY 0000
SP 7E00
PC 700E
AB 703F
MEMO
B7
IR B7
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.99


SPEED
1 9
CC
MC
IC
TRACE
FAST
MAN
ADR
DB ED
COD :
A 00
A' 00
T 00
01000100
SZXHXPNC
00000010
T 00
A.L.U.
B 5B
C 00
D 12
E 34
H 5B
L 04
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 00
Z 00
R 31
I 00
IX 0000
IY 0000
SP 7E00
PC 7007
AB 7030
MEMO
ED
IR ED
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT

Im. 100



Im.101

Pînă aici am citit și interpretat cele 2 coduri de operație ED_H și 42_H.

În ciclul M3 octeții mai puțin semnificativi sînt trecuți prin ALU, rezultatul scăderii L—C înscriindu-se în L (Im 102).

(Observăm și aici graba lui VISIBLE—Z80 care execută M3 în 3 cicluri de tact, în loc de 4 cum se întîmplă de fapt).

Iar apoi este rîndul octeților celor mai semnificativi să viziteze ALU. Rezultatul este : H = H − B. (Im 103).

Remarca 34 : Z80 cunoaște în total 3 tipuri de operații aritmetice pe 16 bit. Iată-le :

ADD	HL, rr	ADD IX, rr	ADD IY, rr
ADC	HL, rr	—	—
SBC	HL, rr	—	—

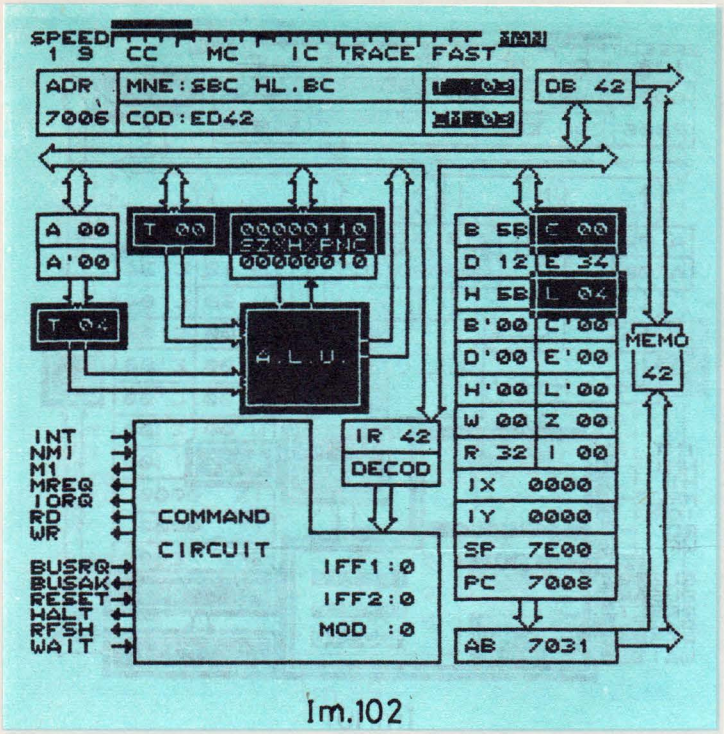
unde rr poate fi :

BC, DE, HL, SP	BC, DE, IX, SP	BC, DE, IY, SP

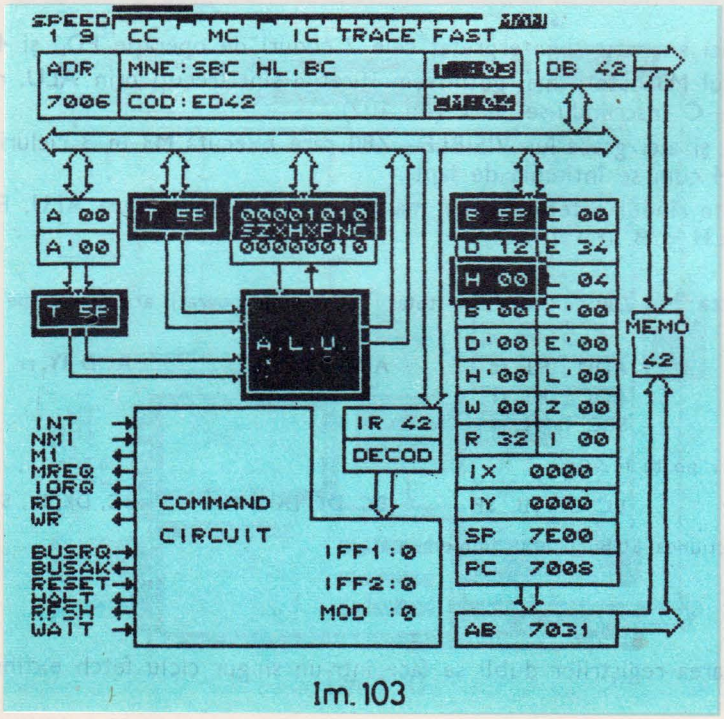
Instrucțiunea SUB nu este implementată.

7008 INC HL Im.104 (IC)

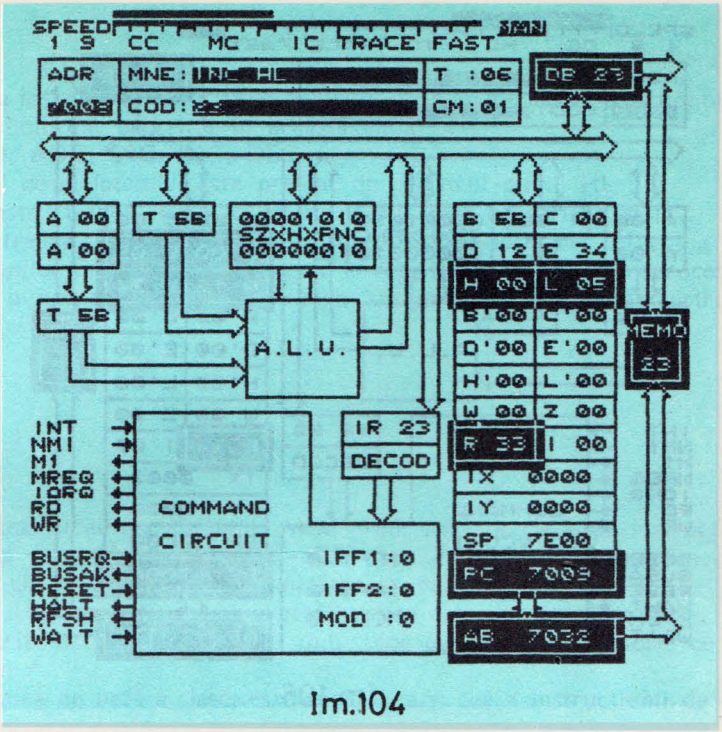
Incrementarea regiștrilor dubli se face într-un singur ciclu fetch extins (CM=1, T=6).


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7006
MNE:SBC HL,BC
COD:ED42
DB 42
A 00
A'00
T 00
00000110
SZ*H*PNC
00000010
B 5B
C 00
D 12
E 34
H 5B
L 04
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 32
I 00
IX 0000
IY 0000
SP 7E00
PC 7008
T 04
A.L.U.
MEMO
42
IR 42
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
AB 7031

Im.102


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7006
MNE:SBC HL,BC
COD:ED42
DB 42
A 00
A'00
T 5E
00001010
SZ*HXPNC
00000010
B 5E
C 00
D 12
E 34
H 00
L 04
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 32
I 00
IX 0000
IY 0000
SP 7E00
PC 7008
T 5F
A.L.U.
MEMO
42
IR 42
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
AB 7031

Im.103



Im.104

Remarca 35 : În simplitatea lor instrucțiunile de incrementare **INC rr**, decrementare **DEC rr** a registrilor dubli sînt foarte democratice, ele putîndu-se aplica tuturor registrilor dubli tip adresă : **BC, DE, HL, IX, IY** și **SP**.

Remarca 36 : Incrementarea, decrementarea registrilor dubli se face independent de **ALU**, folosind același modul electronic care asigură și incrementarea automată a contorului program **PC** în cazul execuției programelor cod mașină.

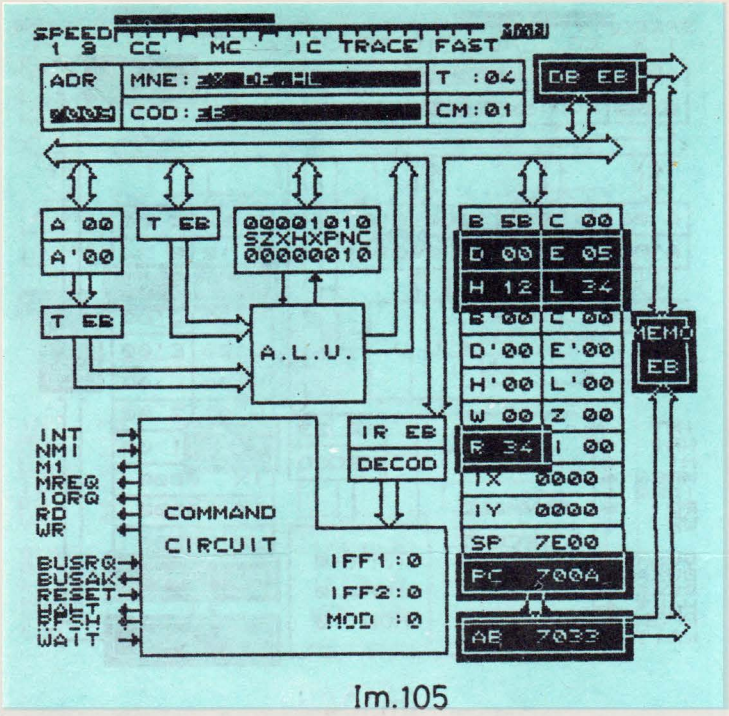
Încheiem aventura a 5-a regrupînd rezultatele,

```
7009        EX      DE,HL              ; Im.105        (IC)
```

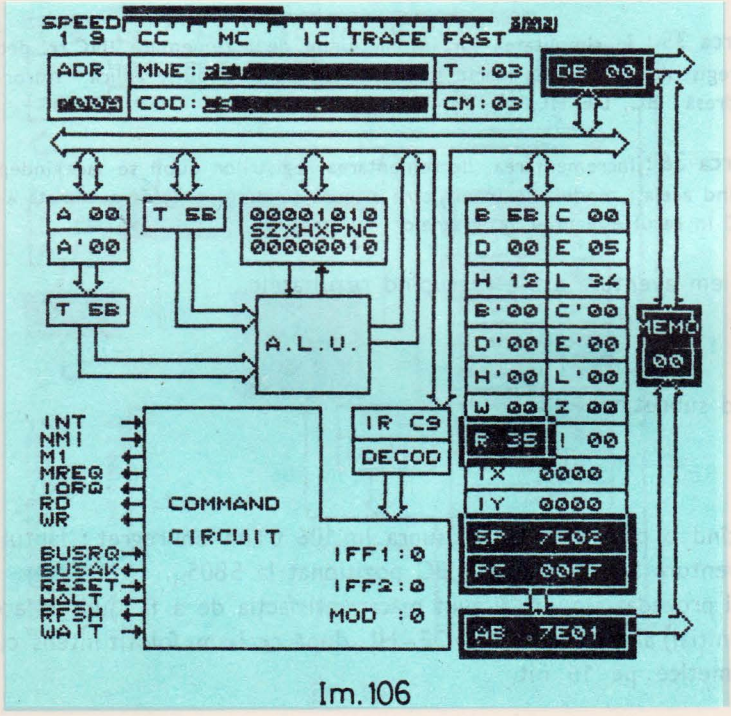
și părăsind subrutina

```
700A        RET                        ; Im.106        (IC)
```

Aruncînd o privire de adio asupra Im.106 trăim un regret : faptul că n-am început aventura 5-a cu registrul **BC** poziționat la $5B05_H$.
Dacă am fi procedat așa, am fi avut măcar satisfacția de a fi ajuns la același conținut (cel inițial) al registrilor **BC, DE, HL**, după ce i-am folosit intens cu instrucțiuni aritmetice pe 16 bit.


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
MNE:
T :04
DB EB
COD:
CM:01
A 00
A'00
T EB
00001010
SZXHXPNC
00000010
A.L.U.
B 5B
C 00
D 00
E 05
H 12
L 34
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 34
I 00
IX 0000
IY 0000
SP 7E00
PC 700A
AB 7033
MEMO
EB
IR EB
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.105


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
MNE:
T :03
DB 00
COD:
CM:03
A 00
A'00
T EB
00001010
SZXHXPNC
00000010
A.L.U.
B 5B
C 00
D 00
E 05
H 12
L 34
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 35
I 00
IX 0000
IY 0000
SP 7E02
PC 00FF
AB 7E01
MEMO
00
IR C9
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.106

6.4. Sinteza clasei ARIT-16

Clasa instrucțiunilor aritmetice pe 16 bit numără **32** de instrucțiuni distincte.
Mnemonicele prezente în această clasă sînt :
ADD, ADC, SBC, INC, DEC
Rolul *acumulatorului* este preluat de registrul dublu **HL**.
La instrucțiunea **ADD** același rol îl pot avea regiștri **IX** și **IY**.
În întreaga clasă se folosește un singur tip de adresare, cel *implicit*, operanzii instrucțiunilor fiind locați obligatoriu în regiștri interni ai microprocesorului **Z80**.
Din punct de vedere funcțional, am împărțit cele 32 de instrucțiuni în 7 grupe.

6.5. Pseudoaventură : instrucțiunile logice pe blocuri de date

Liniștea dinaintea furtunii. Vom trata această clasă de instrucțiuni, fără a recurge la serviciile lui **VISIBLE—Z80** menajîndu-vă ochii înaintea ultimelor 5 aventuri. Un scurt respiro la jumătatea drumului.
Unicul tip de instrucțiune al acestei clase este cel de comparație. În 4 modalități diferite se compară octeții unui bloc de date din memorie cu conținutul registrului A.
Acțiunea de bază a clasei este aceeași ca și cea a instrucțiunii de comparație CP:

$$A - (HL)$$

Rezultatul numeric al scăderii nu se generează, informațiile logice relaționale ($=, >, <$) putîndu-se extrage din registrul indicatorilor de condiție F.
Cele patru mnemonice includ în numele lor inițialele **CP**.
Ori dé cîte ori vom dori să localizăm prima apariție a unui octet dat într-un bloc de date din memorie, vom recurge la serviciile uneia din instrucțiunile clasei LOG—IDR.

6.6. Sinteza clasei LOG—IDR

Totalul instrucțiunilor : 4
Mnemonicele prezente : **CPD, CPI, CPDR, CPIR**
Semnificația mnemonicelor :
CPD — compare & decrement (compară și apoi decrementează)
CPI — compare & increment (compară și apoi incrementează)
CPDR — compare, decrement, repeat (compară, decrementează, repetă)
CPIR — compare, increment, repeat (compară, incrementează, repetă)
Tipuri de adresare folosite :
— **implicit** pentru **A**
— **indirect** pentru octeții blocului de date baleiat.

7

INSTRUCȚIUNI DIVERSE

În capitolele 5 și 6 am avut posibilitatea să realizăm importanța instrucțiunilor aritmetice/logice, și a celor de transfer. Așa cum probabil vă dați seama, ele sînt însă insuficiente pentru implementarea pe calculator a unei probleme oricît de simple. Pentru ca într-un program să se poată lua decizii, în funcție de valoarea instantanee a unui eveniment extern, sau în funcție de valoarea calculată a unei variabile, sînt necesare instrucțiuni de salt. Pentru ca un program să fie cît mai scurt, păstrînd totodată o structură elegantă și inteligibilă, se cer subrutine și instrucțiunile de apel și de revenire din acestea.

Aceste instrucțiuni, care determină structura unui program și dirijează execuția lui, își conjugă eforturile cu alte instrucțiuni aparent mai mărunte și mai puțin semnificative, deservind instrucțiunile aritmetice și logice pentru a realiza obiectivul propus al programului în care au fost incluse. Ele muncesc din plin.

Iar importanța celor ce muncesc, o cunoaștem cu toții.

7.1. Aventura a 6-a : de toate

Pe parcursul căreia vom avea prilejul să vedem în acțiune instrucțiunile aparținînd la 4 clase noi : instrucțiuni de salt, instrucțiuni de apel și revenire din subrutine, instrucțiuni orientate pe bit și instrucțiunile de rotire și deplasare a octeților din regiștri sau celule de memorie.

Iată itinerariul ales.

Programul principal :

```
7000    BIT   7,A
7002    JP    NZ,700A
7005    SCF
7006    RRA
7007    RLCA
7008    JR    C,01
700A    RST   08H
```

Im.107—Im.109 (MC)
Im.110—Im.112 (MC)
Im.113 (MC)
Im.114—Im.119 (CC)
Im.120 (IC)
Im.121—Im.123 (MC)

```
700B    RR    B          ; Im.124—Im.126  (MC)
700D    SRA   B          ; Im.127—Im.129  (MC)
700F    BIT   6,B        ; Im.130         (IC)
7011    CALL  NZ, 7050   ; Im.131—Im.135  (MC)
7014    CCF              ; Im.142         (IC)
7015    CALL  70A0       ; Im.143         (IC)
7018    RES   7,C        ; Im.146         (IC)
701A    HALT             ; Im.147         (IC)
```

Subrutinele :

```
7050    RL    B          ; Im.136         (IC)
7052    RR    C          ; Im.137         (IC)
7054    RES   0,A        ; Im.138         (IC)
7056    RET              ; Im.139—Im.141  (MC)
```

și

```
70A0    RES   7,B        ; Im.144         (IC)
70A2    RET   C          ; Im.145         (IC)
70A3    ADD   A,B        ;
70A4    RET              ;
```

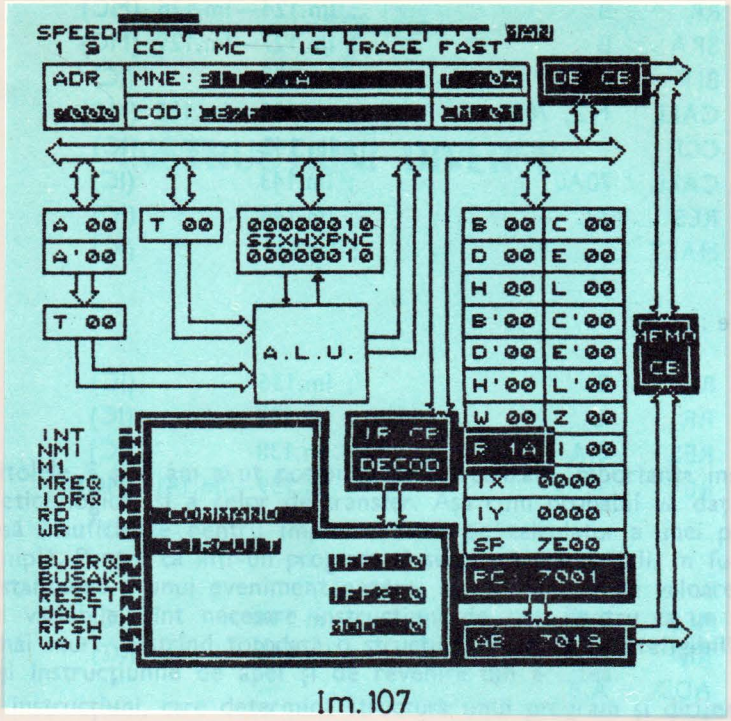
Să pornim la drum pornind de la adresa noastră, adresa 7000H. Fie-ne partener trecător, un bit „1", care ne va însoți în aventură, lăsînd să tresară „inimile" cîtorva regiștri, pentru ca în final să dispară fără urmă, bucuria cunoștințelor acumulate de astădată fiind unica în măsură a ne permite să uităm regretul de-al fi pierdut.

```
7000    BIT   7,A        ; Im.107—Im.109  (MC)
```

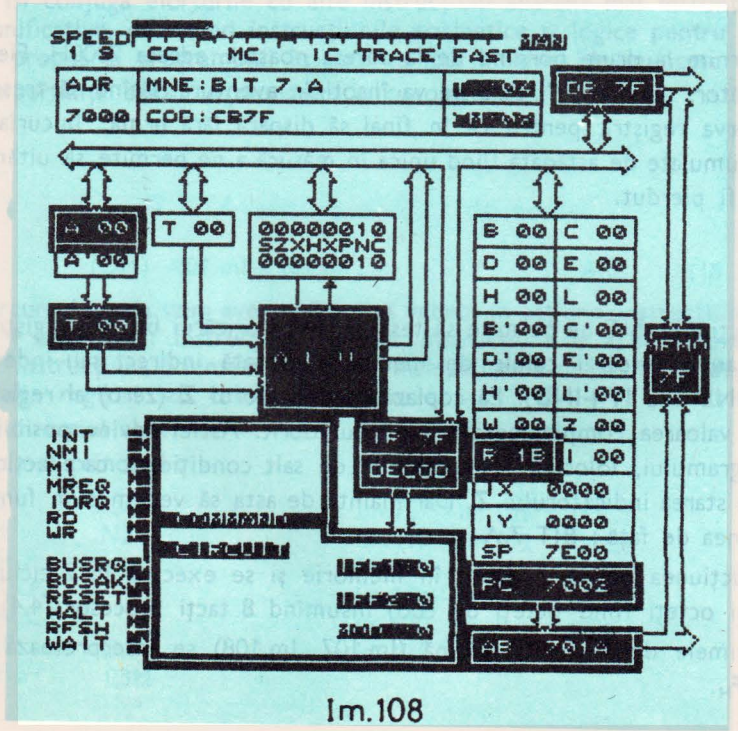
Instrucțiunea BIT ne permite să testăm starea oricărui bit din regiștri de uz general, sau a oricărei celule de memorie adresată indirect sau indexat (prin **HL, IX+IND** sau **IY+IND**). Ea copiază în indicatorul **Z** (zero) al registrului de flaguri F, valoarea complementată a bitului dorit. Astfel devine posibilă ramificarea programului, folosind instrucțiunile de salt condiționat care acționează în funcție de starea indicatorului **Z**. Dar înainte de asta să vedem cum funcționează instrucțiunea de față : **BIT 7,A**.

Instrucțiunea ocupă 2 octeți în memorie și se execută în 2 cicluri mașină **M1** (ambii octeți fiind octeți de cod) însumînd 8 tacți procesor (4,4).

În primele două cicluri mașină (Im.107, Im.108) se interpretează codurile CB_H și $7F_H$.

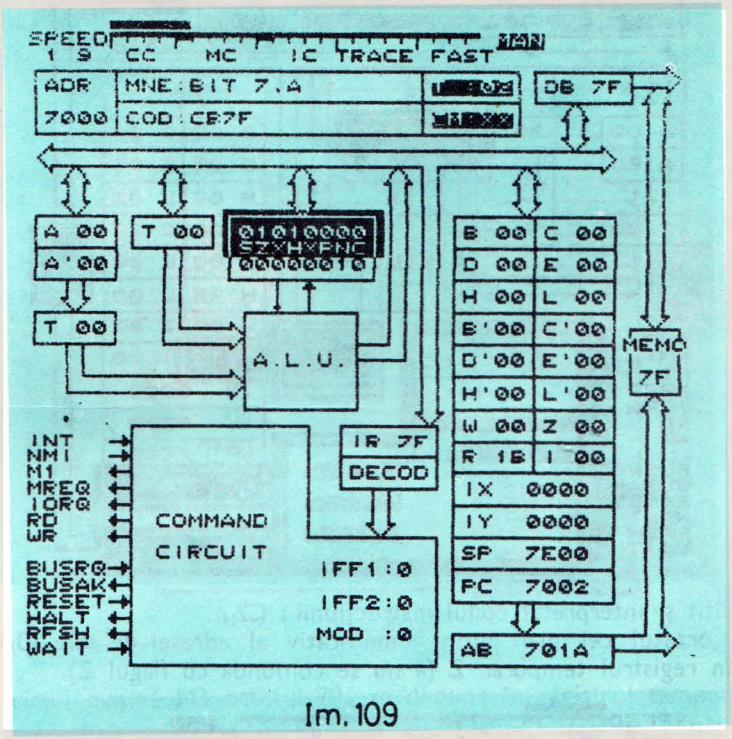

SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
MNE:
COD:
DE CB
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 1A
I 00
IX 0000
IY 0000
SP 7E00
PC 7001
AB 7019
IR CB
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.107


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:BIT 7,A
COD:CB7F
DE 7F
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 1B
I 00
IX 0000
IY 0000
SP 7E00
PC 7002
AB 701A
IR 7F
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.108

Urmează ciclul suprapus în care instrucțiunea este executată fizic (CM=XX T=2). Înainte de a-l vedea rememorăm că **Z=0** și **bit$_7$** al registrului **A** este și el 0.



Im.109

După execuția ultimei secvențe din instrucțiune (Im.109) **Z=1 = $\overline{\text{bit}_7\text{A}}$.**

Remarca 37 : Instrucțiunea **BIT** ar fi putut copia desigur și bitul real (nu complementul lui). Această inversare este necesară pentru a corela acțiunea instrucțiunii cu semantica mnemonicelor instrucțiunilor de salt condiționat.

Cînd spunem "*jump if not zero*" („*salt dacă diferit de zero*") ne referim nu la conținutul flagului **Z** ci la semnificația conținutului lui. Într-adevăr, starea **Z=1** înseamnă că elementul testat (octetul din registrul **A** sau un bit selectat) este egal cu 0.

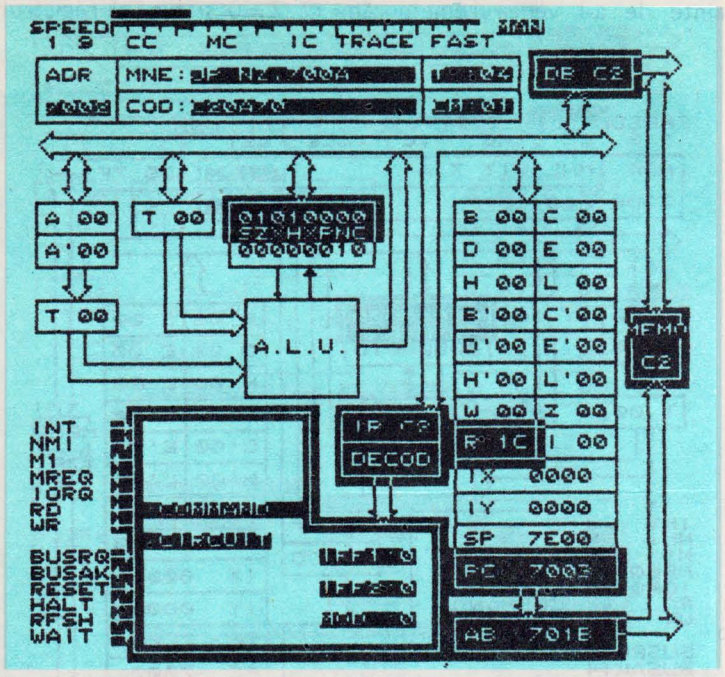
Să testăm deci condiția aleasă :

```
7002    JP      NZ, 700A        ; Im.110—Im.112 (MC)
```

Instrucțiunea de față este una din cele 8 instrucțiuni de salt condiționat absolut, care folosesc adresarea directă. Dacă condiția testată **NZ**, este adevărată atunci se efectuează saltul la adresa specificată în enunțul instrucțiunii. În caz contrar execuția programului continuă cu instrucțiunea următoare.

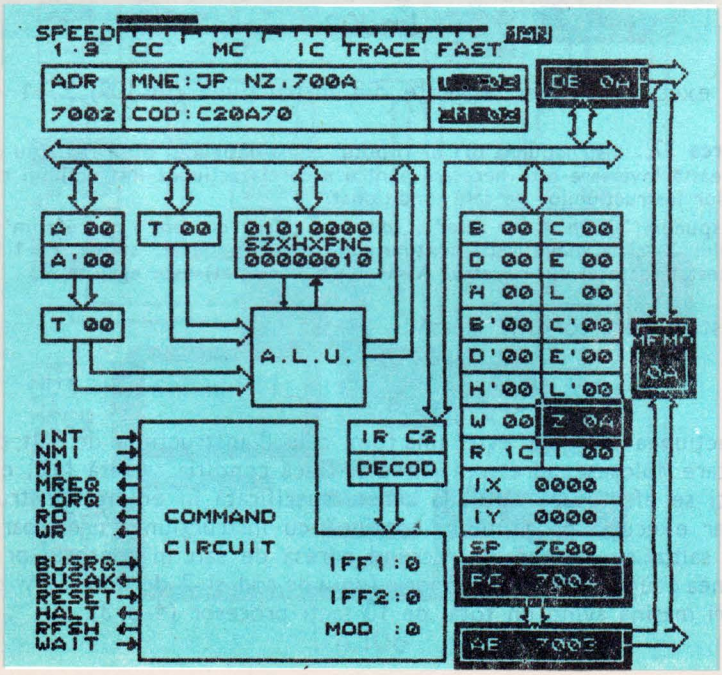
Fizic, saltul se efectuează înscriind adresa de salt în contorul program **PC**. Instrucțiunea ocupă 3 octeți în memorie (unul de cod și 2 de adresă) și se execută în 3 cicluri mașină avînd un total de 10 tacți procesor (4, 3, 3).

Să le vedem :



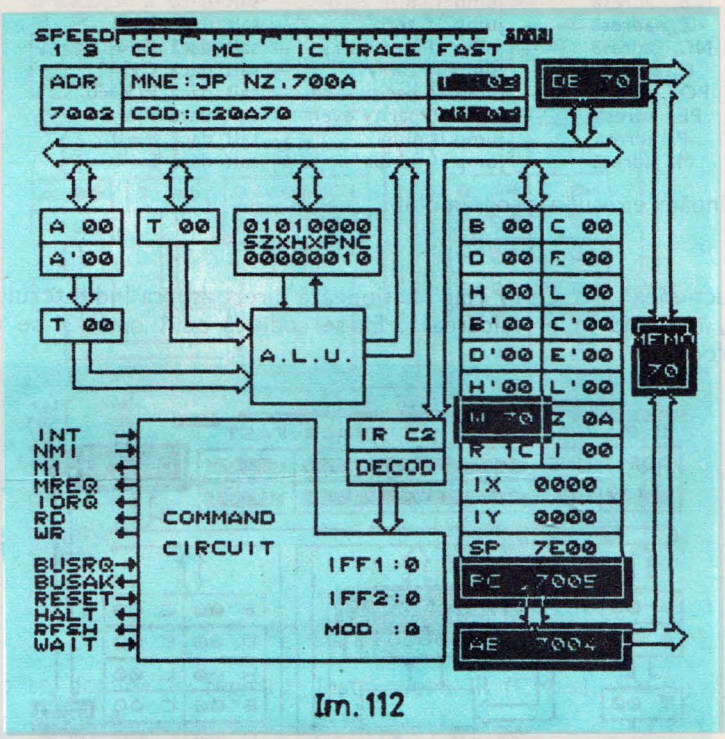
Im.110

S-a citit şi interpretat codul instrucţiunii : $C2_H$.
Urmează octetul cel mai puţin semnificativ al adresei de salt ($0A_H$) care se depune în registrul temporar Z (a nu se confunda cu flagul Z).



Im.111

După care se citește octetul cel mai semnificativ al adresei de salt :



Im. 112

Citit în ciclul mașină **M3** octetul 70_H se depune în registrul temporar **W**.

Cum condiția stipulată în enunțul instrucțiunii (salt dacă diferit de zero) nu este adevărată — bit_7 din **A** fiind 0 — saltul nu se efectuează, execuția programului continuînd cu instrucțiunea următoare, locată la adresa 7005_H (valoarea curentă a contorului program **PC**).

Remarca 38 : Reținem faptul că la sfîrșitul ciclului fetch (M1), microprocesorul „știa" deja dacă saltul va fi efectuat sau nu. Dacă saltul urma să nu fie efectuat (cazul de față), citirea celor 2 octeți de adresă (ciclurile **M2** și **M3**), nu mai este necesară, instrucțiunea putînd fi executată prin dubla incrementare a contorului program **PC**, astfel încît el să indice adresa de început a instrucțiunii următoare. Astfel s-ar fi cîștigat timp — cel puțin 4 cicluri de tact ceea ce reprezintă 40%. **Z80** nu procedează însă așa, preluînd această insuficiență de la predecesorul său microprocesorul **I8080** — tocmai pentru a fi compatibil de sus în jos cu acesta. Insuficiența a fost însă eliminată la implementarea instrucțiunilor de salt relativ care execută un număr diferit de cicluri de tact pentru condiția adevărată sau falsă, instrucțiuni pe care **I8080** nu le posedă. Vom vedea.

Remarca 39 : Instrucțiunile de salt condiționat absolut, testează cei 4 indicatori de condiție principali : **S** — semn, **Z** — zero, **P/V** — paritate/depășire și **Cy** — transport. Pentru fiecare flag testat condiția de salt se poate enunța în 2 moduri :

— salt dacă flagul este 1
— salt dacă flagul este 0

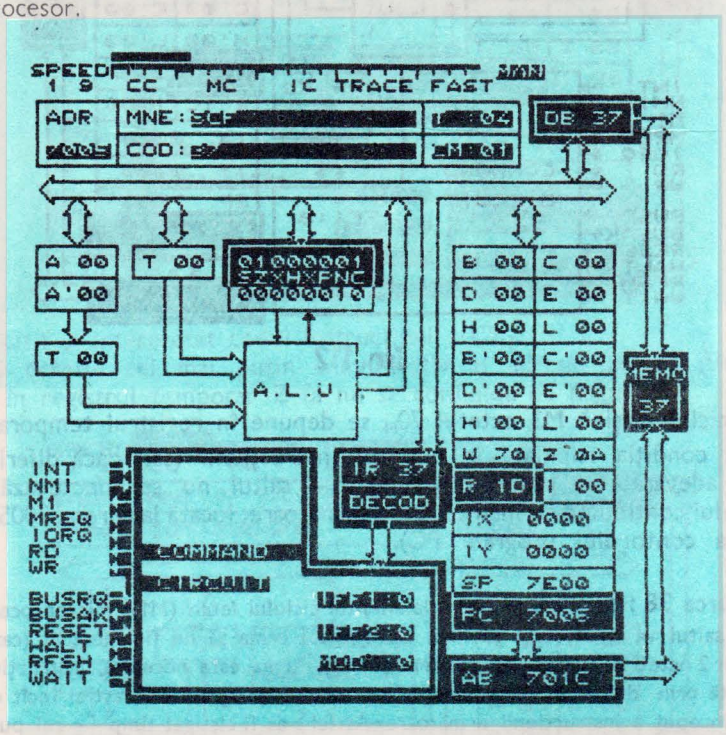
Rezultă cele 8 instrucțiuni amintite :

JP	NZ, adresă	— jump if not zero	— salt dacă	Z=0
JP	Z, adresă	— jump if zero	— salt dacă	Z=1
JP	NC, adresă	— jump if not carry	— salt dacă	Cy=0
JP	C, adresă	— jump if carry	— salt dacă	Cy=1
JP	PO, adresă	— jump if parity odd	— salt dacă	P/V=0
JP	PE, adresă	— jump if parity even	— salt dacă	P/V=1
JP	P, adresă	— jump if plus	— salt dacă	S=0
JP	M, adresă	— jump if minus	— salt dacă	S=1

Continuăm execuția programului :

```
7005    SCF                     ; Im.113        (MC)
```

Instrucțiunea „Set Carry Flag" acționează direct asupra indicatorului de transport **Cy**, înscriind în el valoarea 1. Ea se codifică pe 1 octet și se execută în 4 tacți procesor.



Im.113

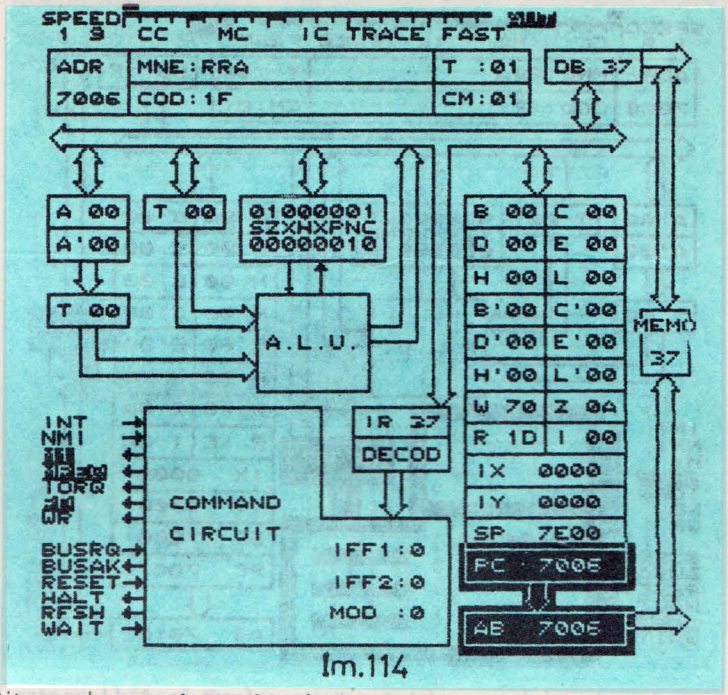
Remarca 40 : Instrucțiunile SCF și CCF (Complement Carry Flag) sînt singurele instrucțiuni care permit manipularea directă a unui bit indicator de condiție. Ele acționînd asupra unui bit individual, au fost incluse în clasa instrucțiunilor orientate pe bit (BITSR) alături de BIT, SET și RES.

```
7006    RRA                     ; Im.114—Im.119 (CC)
```

RRA este una din cele 4 instrucțiuni de rotire a conținutului registrului A pe care le posedă și microprocesorul **I8080**. **Z80** mai adaugă 70 de instrucțiuni de acest gen incluse de noi în clasa instrucțiunilor de rotire/deplasare ROT/SHIFT.

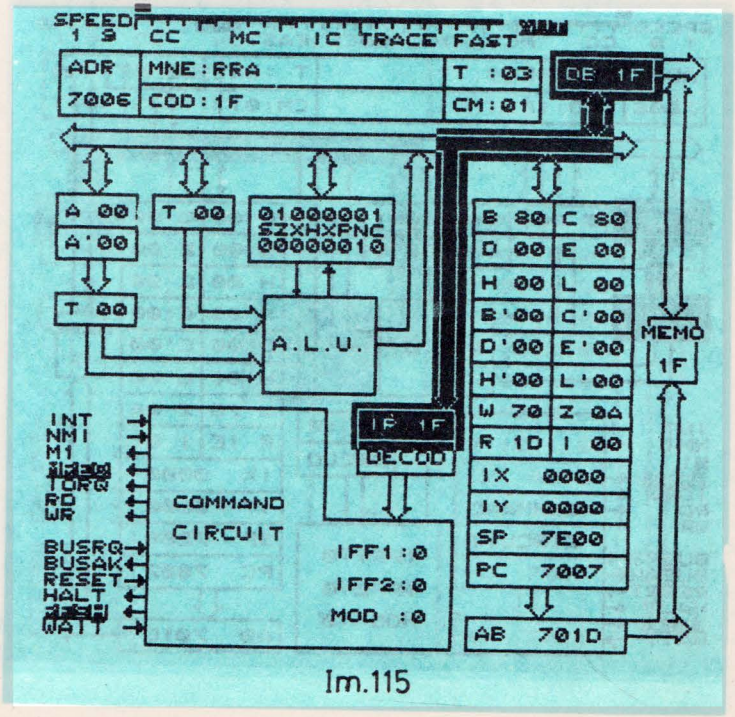
Să urmărim execuția instrucțiunii RRA (Rotate Right Accumulator), tact cu tact.

Începe ciclul fetch CM=1, T=1)



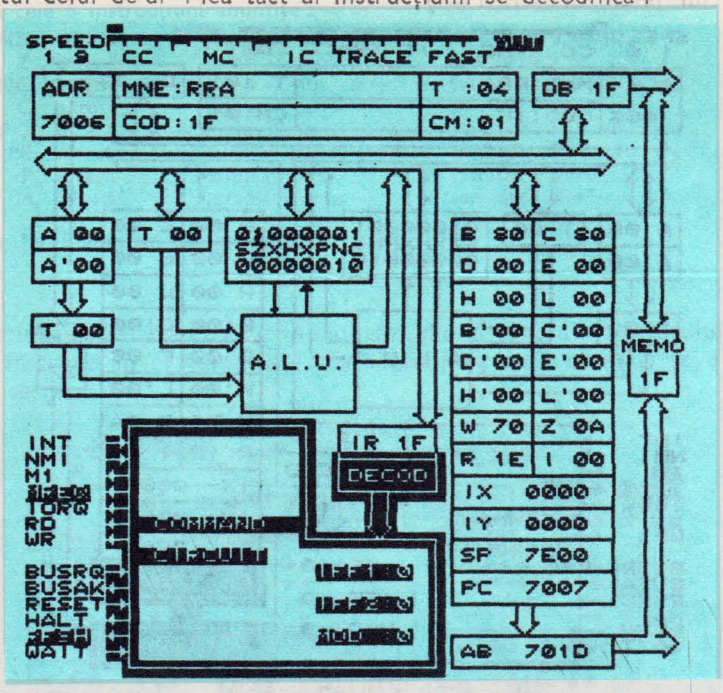
Im.114

Codul citit se depune în registrul instrucțiune IR (CM=1, T=3) :



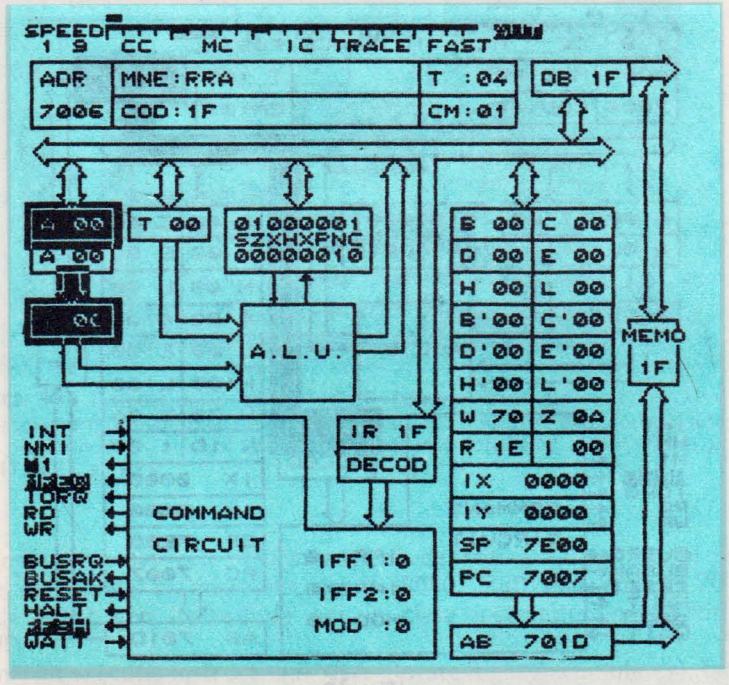
Im.115

La începutul celui de-al 4-lea tact al instrucțiunii se decodifică :



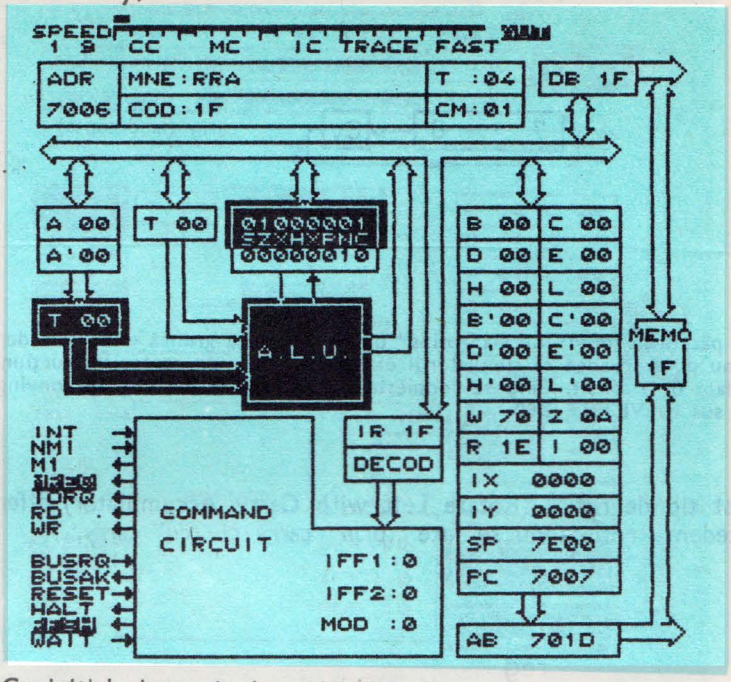
Im. 116

Apoi conținutul registrului acumulator este transferat în **ALU** pentru a fi rotit :



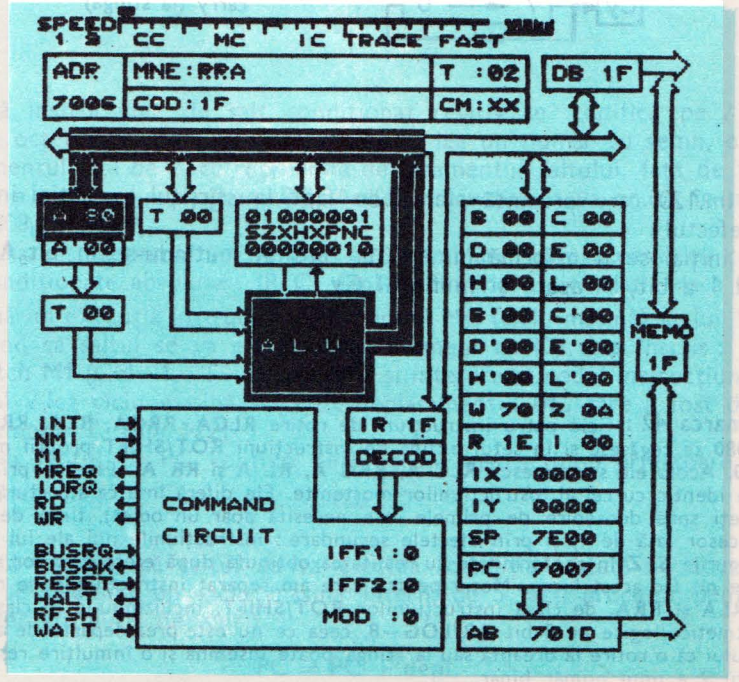
Im. 117

Rotate Right : rotirea se face la dreapta, prin „**carry**", astfel încît bit_0 al lui A se transferă în Cy,



Im.118

iar bitul Cy inițial ajunge în bit_7 al lui A.



Im.119

Remarca 41 : Acțiunea instrucțiunii este deci :

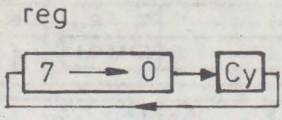


Fig. 7.1. Rotirea „prin" carry (la dreapta)

Din păcate în Im.119 nu s-a „prins" ultima secvență grafică executată de **VISIBLE Z80** și astfel nu putem vedea că simulatorul execută într-adevăr corect instrucțiunea, flagul Cy fiind de fapt 0 la sfîrșit. Cine se îndoiește de asta n-are decît să se convingă rulînd secvența de sus cu **VISIBLE Z80.**

Acest tip de rotire (Rotate Left with Carry Accumulator) diferă puțin de cel precedent : rotirea nu se face *„prin"* carry ci *„cu"* carry.

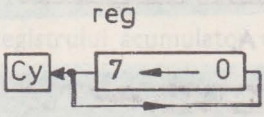
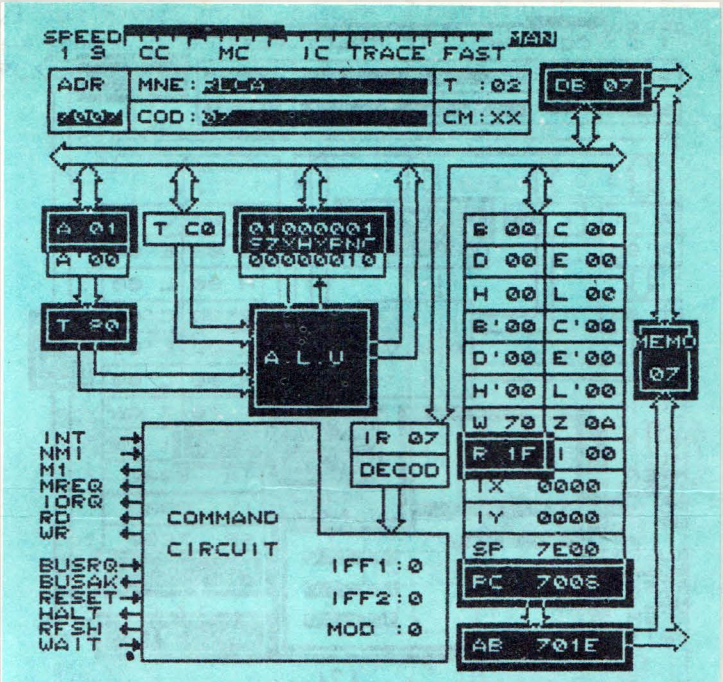


Fig. 7.2. Rotirea „cu" carry (la stînga)

În Im.120 care prezintă starea de fapt la sfîrșitul execuției instrucțiunii, redăm efectul :
$A=80_H$ inițial se transformă în $A=01_H$, bit_7 A mutîndu-se în bit_0A. Totodată valoarea 1 a bit_7 A apare și în flagul **Cy**.

Remarca 42 : Cele patru instrucțiuni de rotire **RLCA, RRCA, RLA, RRA** moștenite de la **I8080** se regăsesc și în setul extins de instrucțiuni **ROT/SHIFT** proprii microprocesorului **Z80**. Acolo ele se numesc : **RLC A, RRC A, RL A** și **RR A**. Efectul primar al acestora este identic cu cel al instrucțiunilor moștenite. Ele diferă însă ca structură (se codifică pe 2 octeți spre deosebire de primele care necesită doar un octet), timp de execuție (8 tacți procesor față de 4) și prin efectele secundare : instrucțiunile noi ale lui **Z80** poziționează flagurile **S, Z** în conformitate cu realitatea obținută după efectuarea lor, pe cînd cele moștenite nu fac acest lucru. Motiv pentru care am separat instrucțiunile de rotire **RLCA, RRCA, RLA** și **RRA**, de clasa instrucțiunilor **ROT/SHIFT**, incluzîndu-le în clasa instrucțiunilor aritmetice/logice pe 8 bit **AR/LOG—8**, ceea ce nu este prea departe de adevăr, datorită faptului că o rotire la dreapta sau la stînga poate însemna și o înmulțire respectiv o împărțire cu 2 a unui număr binar.



Im.120

Această instrucțiune de salt condiționat relativ se codifică pe 2 octeți : primul este octetul de cod (38_H) iar cel de-al 2-lea un număr cu semn, exprimat în complementul față de 2, ce reprezintă deplasamentul saltului, față de valoarea instantanee a contorului program PC. Astfel instrucțiunea va ocupa în memorie adresele 7008_H și 7009_H.

Structura instrucțiunii, în limbaj de asamblare, este aceeași cu cel de la salturile condiționate absolute : **JR C, depl** însemnînd „salt relativ dacă Cy=1".

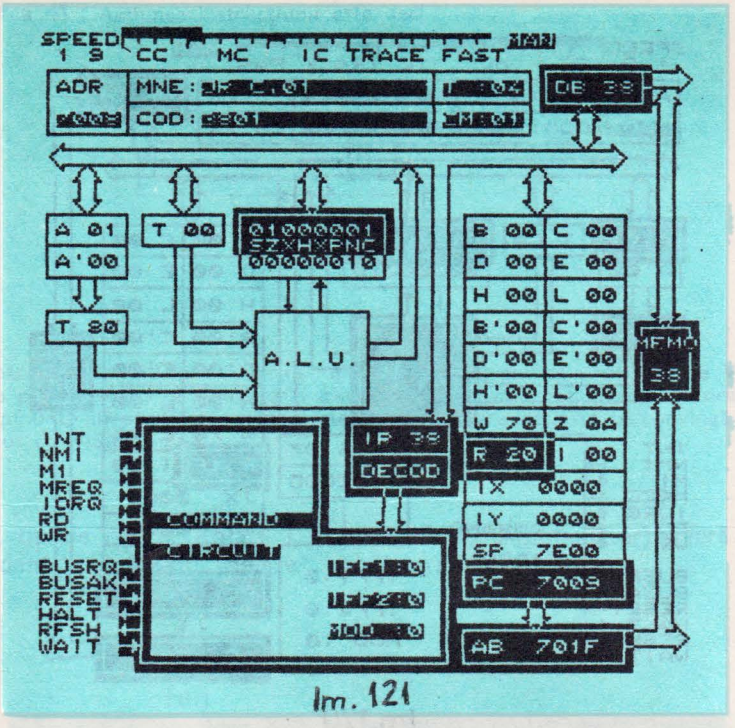
Să urmărim execuția instrucțiunii, în modul **MC** (ciclu mașină) al lui **VISIBLE — Z80,** știind că saltul se va efectua, din moment ce **Cy** este înscris :
În ciclul fetch **M1** (CM=1, T=4) am citit și interpretat codul instrucțiunii : 38_H.
În cel de-al 2-lea ciclu mașină s-a citit deplasamentul 01_H care a fost depus în registrul tampon **T**. (Im. 122)

Reținem faptul că PC conține deja adresa instrucțiunii următoare :

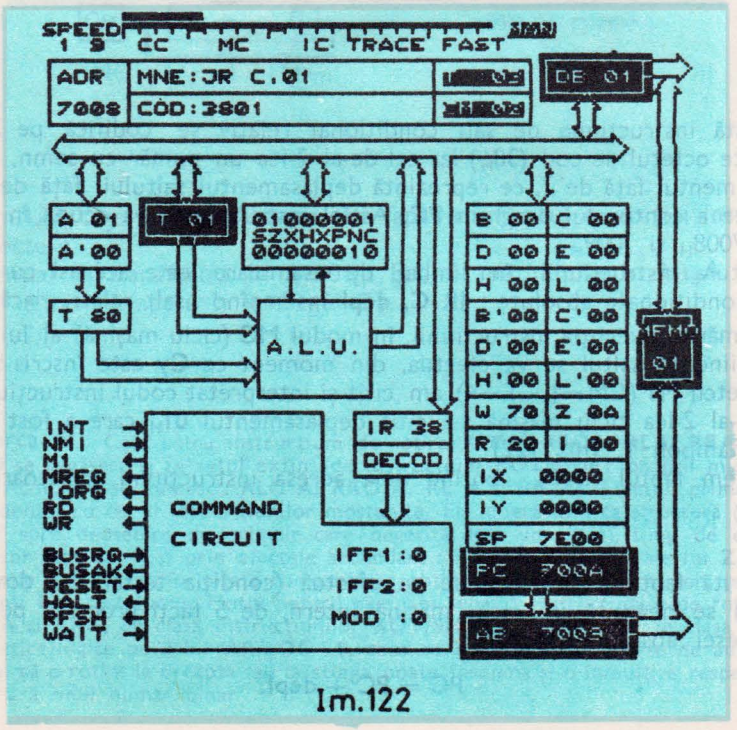
```
700A        RST 08H
```

Datorită faptului că saltul se va efectua (condiția testată s-a dovedit a fi adevărată) se necesită un ciclu mașină intern, de 5 tacți procesor pentru calculul adresei absolute de salt :

$$PC = PC + depl.$$

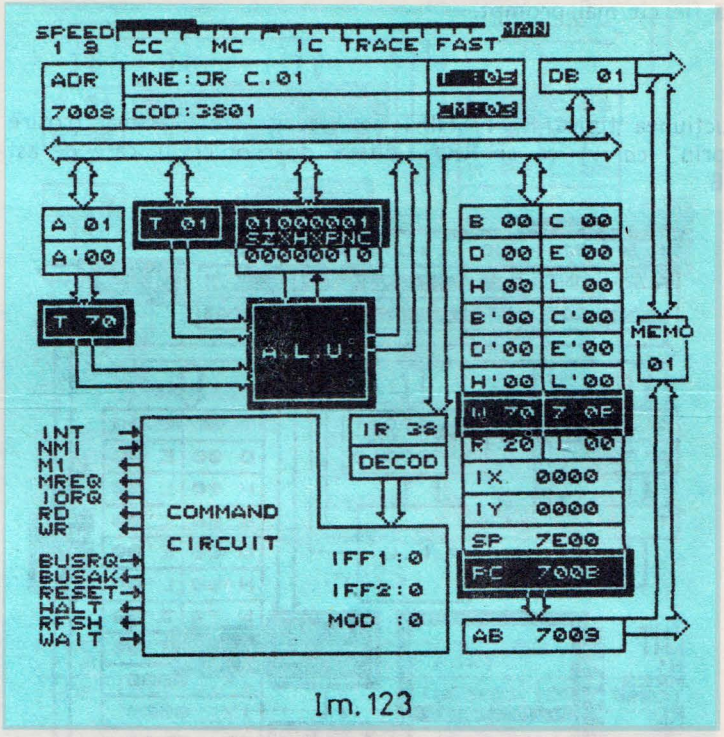

SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
MNE :
COD :
DB 38
A 01
A' 00
T 00
01000001
SZXHXPNC
00000010
T 80
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 70
Z 0A
R 20
I 00
IX 0000
IY 0000
SP 7E00
PC 7009
AB 701F
MEMO
38
IR 38
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT

Im. 121


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7008
MNE : JR C,01
COD : 3801
DB 01
A 01
A' 00
T 01
01000001
SZXHXPNC
00000010
T 80
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 70
Z 0A
R 20
I 00
IX 0000
IY 0000
SP 7E00
PC 7008
AB 7009
MEMO
01
IR 38
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1 : 0
IFF2 : 0
MOD : 0

Im.122

Din Im.123, care fixează ultima „mișcare" a instrucțiunii, observăm că rezultatul adunării s-a generat în primă instanță în registrul dublu **WZ**, fiind apoi transferat în **PC**. Saltul se face deci la $700B_H$, ocolind instrucțiunea de la $700A_H$.



Im. 123

Remarca 43 : Dacă condiția de salt n-ar fi fost satisfăcută, atunci nici cel de-al 3-lea ciclu mașină **M3**, n-ar fi fost necesar, fiindcă **PC** indica deja adresa de început a instrucțiunii următoare ($700A_H$). Execuția durează **12** tacți procesor **(4, 3, 5)** dacă se efectuează saltul, și **7** tacți procesor **(4, 3)** dacă nu.

Remarca 44 : Instrucțiunile de salt condiționat relativ nu testează decît 2 indicatori de condiție, pe cei importanți, totalul for fiind deci 4 :

JR	**NZ**, deplasament	— jump relative if not zero	— salt relativ dacă **Z**=0
JR	**Z**, deplasament	— jump relative if zero	— salt relativ dacă **Z**=1
JR	**NC**, deplasament	— jump relative if not carry	— salt relativ dacă **Cy**= =0
JR	**C**, deplasament	— jump relative if carry	— salt relativ dacă **Cy**= =1

Remarca 45 : Plaja adreselor care se poate acoperi prin salturi relative este de [+127, —128] dacă se consideră ca punct de referință adresa de început a instrucțiunii care succede instrucțiunea de salt relativ.

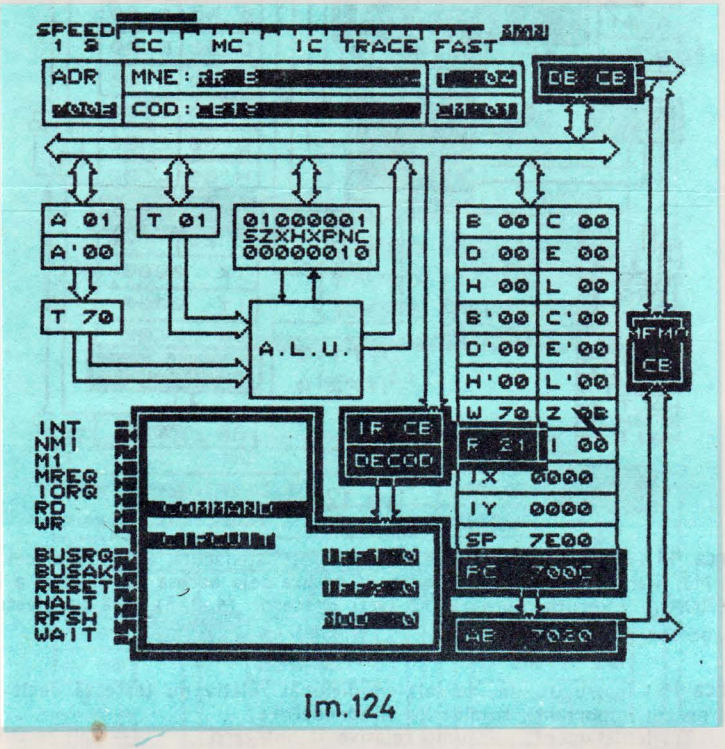
Considerînd adresa de început a instrucțiunii de salt ca punct de referință, plaja salturilor posibile devine [+129, —126].

Sărind deci peste instrucțiunea **RST 08H** vom continua călătoria la adresa $700B_H$ nu înainte de a ști că instrucțiunile **RST** (8 la număr) sînt instrucțiuni de apel a unor subrutine locate la adrese fixe de memorie ($0,8_H$, 10_H, ..., 38_H). Adre-

sele fiind fixe, ele nu trebuie specificate sub forma a doi octeți în cîmpul instrucțiunii, ci adresarea se poate face implicit, în codul de operație. Reducînd numărul de octeți necesari de la 3 la 1, instrucțiunile **RST** devin foarte rapide, folosirea lor fiind recomandată la acceptarea unei cereri de întrerupere, unde răspunsul trebuie să fie cît mai prompt.

700B RR B Im.124 Im.126 (MC)

Instrucțiunea tipică **ROT/SHIFT** are același efect primar (rotire la dreapta (right) „prin" *carry*) ca și **RRA**, diferă doar obiectul, care de astă dată este registrul **B**.

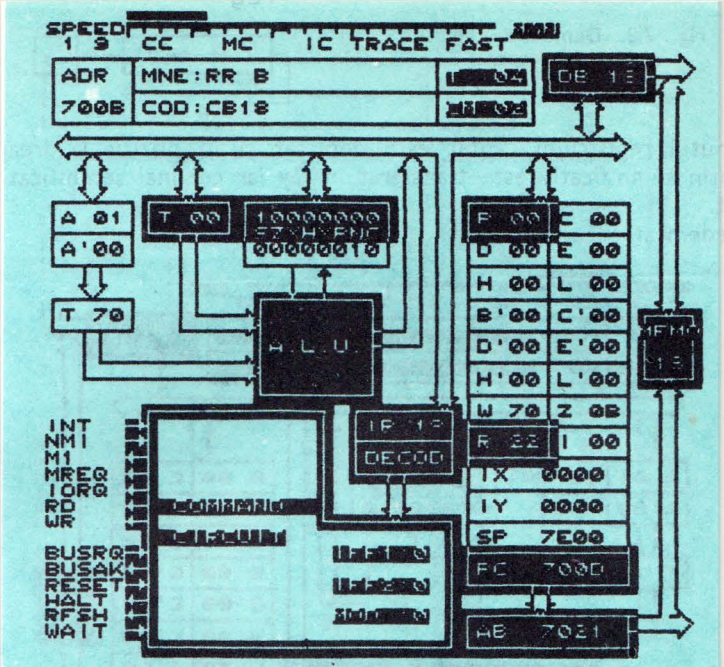


Im.124

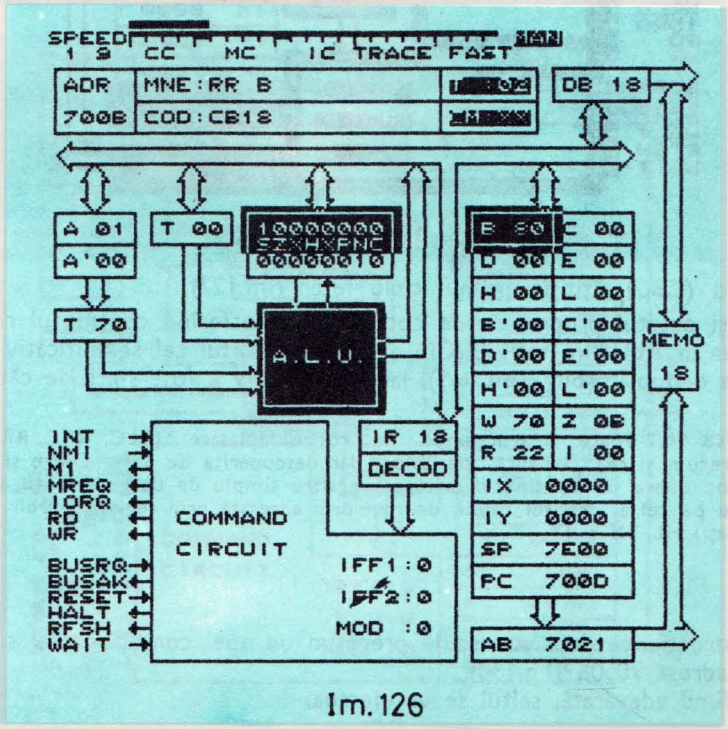
Primul ciclu fetch (Im.124) citește codul caracteristic al clasei ROT/SHIFT, CB_H. Cel de-al doilea ciclu mașină (de asemenea fetch) citește codul propriu-zis (Im.125). Execuția propriu-zisă a instrucțiunii, avînd loc într-un tact ascuns în ciclul mașină al instrucțiunii următoare (CM=XX, T=2)
(Iată-l pe camaradul nostru, acel plăpînd bit „1", de care aminteam la începutul aventurii, apărînd în „viața" registrului **B** : $bit_7B=1$)

700D SRA B Im.127 Im.129 (MC)

SRA este o instrucțiune de deplasare tipică pentru **Z80.** Semnificația mnemonicei alese (**S**hift **R**ight **A**rithmetic) poate fi contestată, efectul în schimb nu : (vezi fig. 7.3).


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
MNE:RR B
7008
COD:CB18
DB 18
A 01
A'00
T 00
10000000
SZ-H-PNC
00000010
T 70
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 70
Z 0B
R 22
I 00
IX 0000
IY 0000
SP 7E00
PC 700D
AB 7021
IR 18
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.125


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
MNE:RR B
7008
COD:CB18
DB 18
A 01
A'00
T 00
10000000
SZ-H-PNC
00000010
T 70
A.L.U.
B 00
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 70
Z 0B
R 22
I 00
IX 0000
IY 0000
SP 7E00
PC 700D
AB 7021
MEMO
18
IR 18
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.126

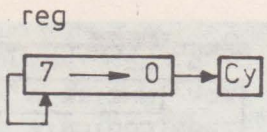
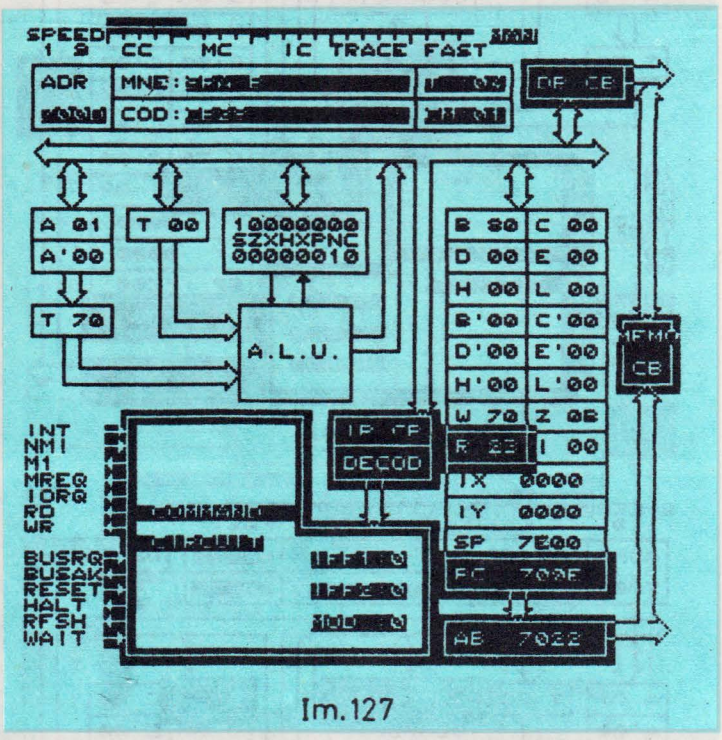


Fig. 7.3. Deplasarea aritmetică (la dreapta)

Conținutul registrului căutat este deplasat cu o poziție la dreapta : bitul cel mai puțin semnificativ este transferat în Cy iar cel mai semnificativ, rămîne neschimbat.

S-o vedem și pe asta :



Im.127

Același cod (CB_H) citit în primul ciclu fetch (Im.127)

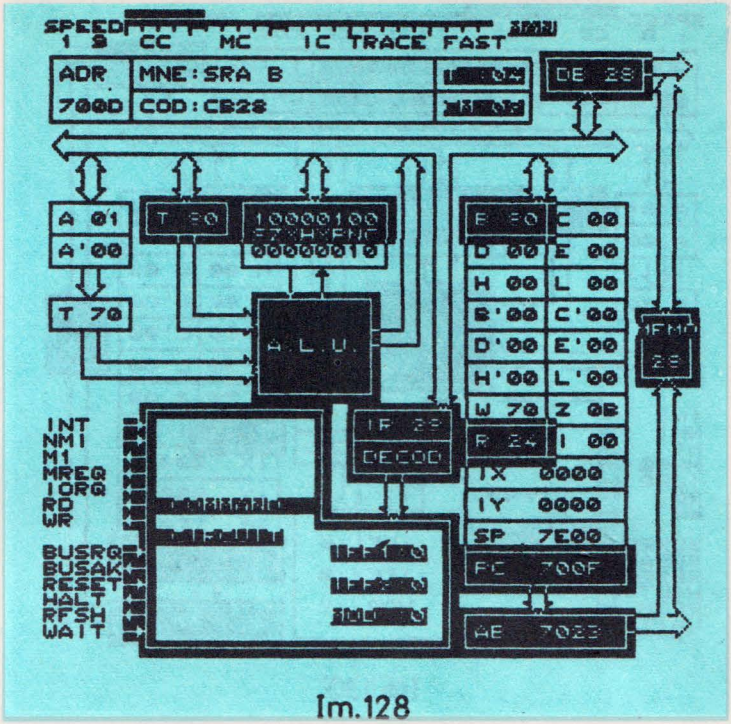
Se citește și octetul propriu-zis de cod (28_H), transferînd conținutul registrului **B** la intrarea în **ALU**. În Im.129, (CM=XX, T=2) bitul cel semnificativ al registrului **B** s-a dedublat ($bit_6 = bit_7 = 1$), iar flagul **carry** a fost șters de către $bit_0 B$.

Remarca 46 : Toate instrucțiunile de rotire/deplasare **(RLC, RRC, RR, RL, SLA, SRA, SRL** precum și cea „ascunsă" de Zilog, dar descoperită de Szasz Detre și notată de noi **SLH)** pot opera pe conținutul oricărui registru simplu de uz general **(B, C, D, E, H, L** sau **A)** sau pe cel al oricărei celule de memorie adresată prin regiștri dubli privilegiați (mereu aceeași) **HL, IX** și **IY.**

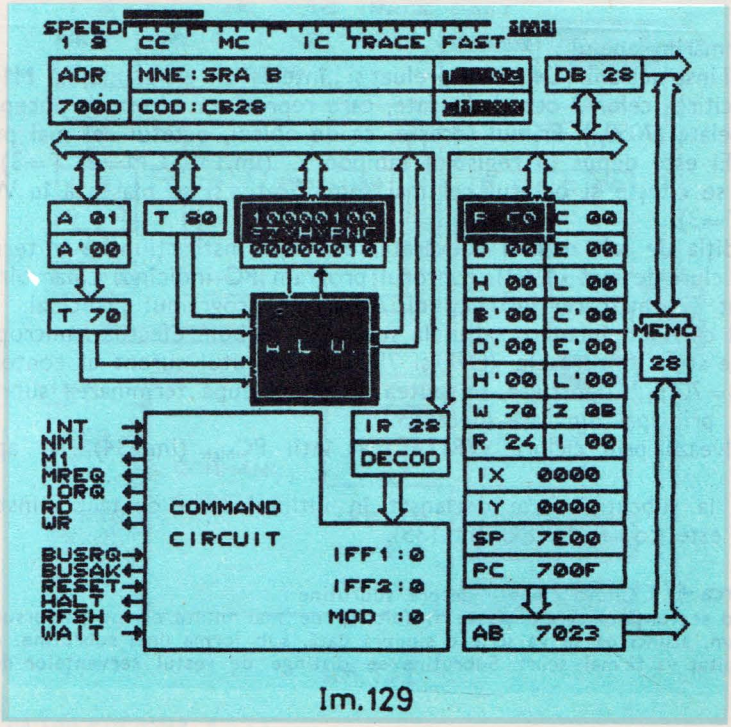
```
700F    BIT    6,B          ;Im.130        (IC)
```

Cu instrucțiunea deja cunoscută pregătim un apel condiționat al subrutinei locate la adresa 7050_H (Im.130).

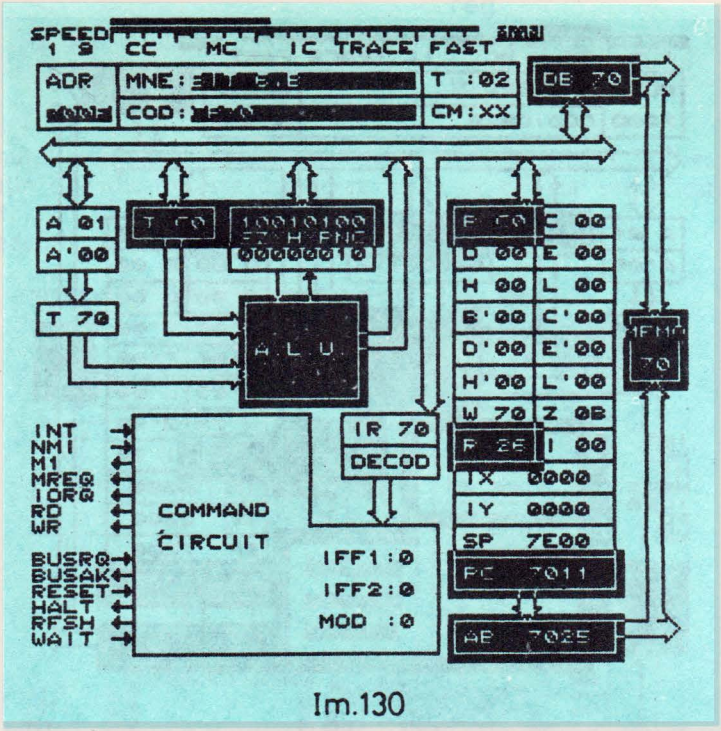
Condiția fiind adevărată, saltul se va efectua.


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7000
MNE: SRA B
COD: CB28
DB 28
A 01
A' 00
T 80
10000100
SZXHXPNC
00000010
T 70
A.L.U.
F 80
C 00
D 00
E 00
H 00
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 70
Z 0B
R 24
I 00
IX 0000
IY 0000
SP 7E00
PC 700F
AB 7023
MEMO
28
IR 28
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.128


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7000
MNE: SRA B
COD: CB28
DB 28
A 01
A' 00
T 80
10000100
SZXHXPNC
00000010
T 70
A.L.U.
F C0
C 00
D 00
E 00
H 00
L 00
B' 00
C' 00
D' 00
E' 00
H' 00
L' 00
W 70
Z 0B
R 24
I 00
IX 0000
IY 0000
SP 7E00
PC 700F
AB 7023
MEMO
28
IR 28
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1 : 0
IFF2 : 0
MOD : 0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

Im.129



Im.130

7011 CALL NZ, 7050 :Im.131—Im.135 (MC)

Să urmărim **apelul :**

Codul instrucțiunii ($C4_H$) este preluat și „înțeles" în ciclul mașină **M1** (Im.131). Urmează citirea **celor 2 octeți** de date, care reprezintă adresa de început a subrutinei apelate (7050_H). **Primul** sosește, ca de obicei, octetul cel mai puțin semnificativ. El este depus în registrul tampon **Z**. (Im.132, CM=2, T=3).

Apoi se citește și **octetul** cel mai semnificativ și se plasează în **W** (Im.133, CM=3, T=3).

Dacă condiția de **salt n-ar fi** adevărată, execuția instrucțiunii s-ar termina aici, după 10 cicluri de **tact (4,3,3)**, contorul program **PC** indicînd, chiar bine, adresa de început a **următoarei instrucțiuni** 7014_H din programul principal.

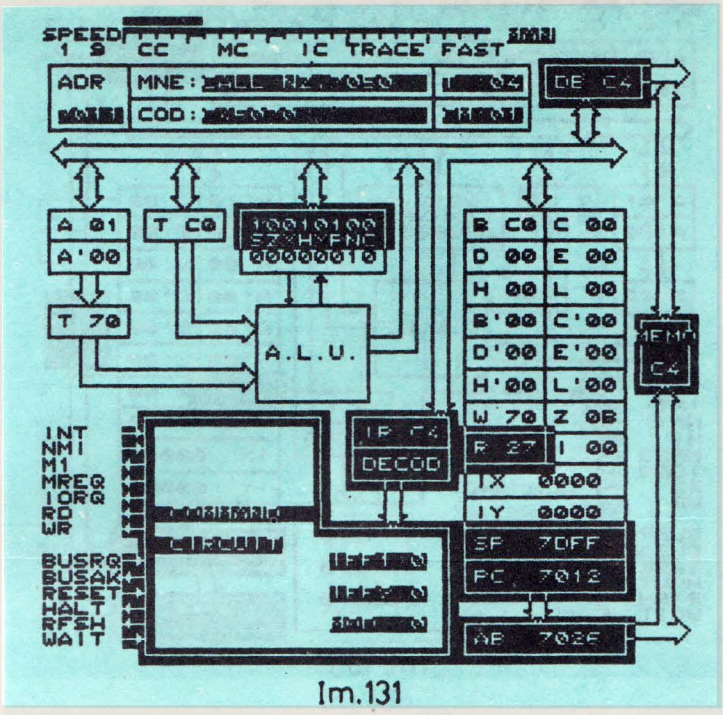
Știind că n-are încotro, saltul la subrutină trebuie efectuat, microprocesorul salvează pe stivă (la adresele 7DFF și 7DFE) conținutul curent al contorului program ($PC=7014_H$), pentru a se putea întoarce după terminarea subrutinei, în programul principal din care plecase.

Se salvează prin cicluri **WRITE** mai întîi PC_{high} (Im.134), iar apoi PC_{low} (Im.135).

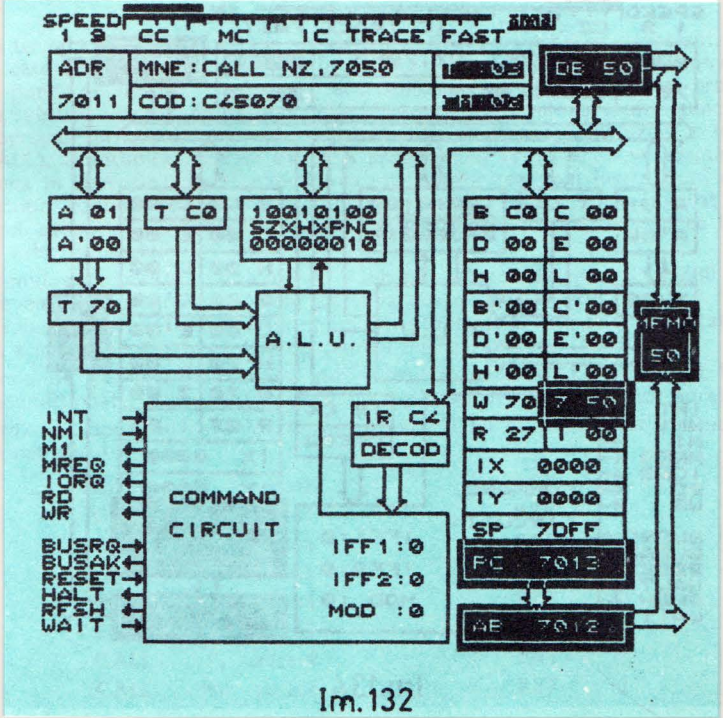
Saltul la subrutină este declanșat în ultimul ciclu de tact a instrucțiunii, cînd **WZ** este copiat în **PC** (Im.135).

Remarca 47 : Cîteva cuvinte despre subrutine :

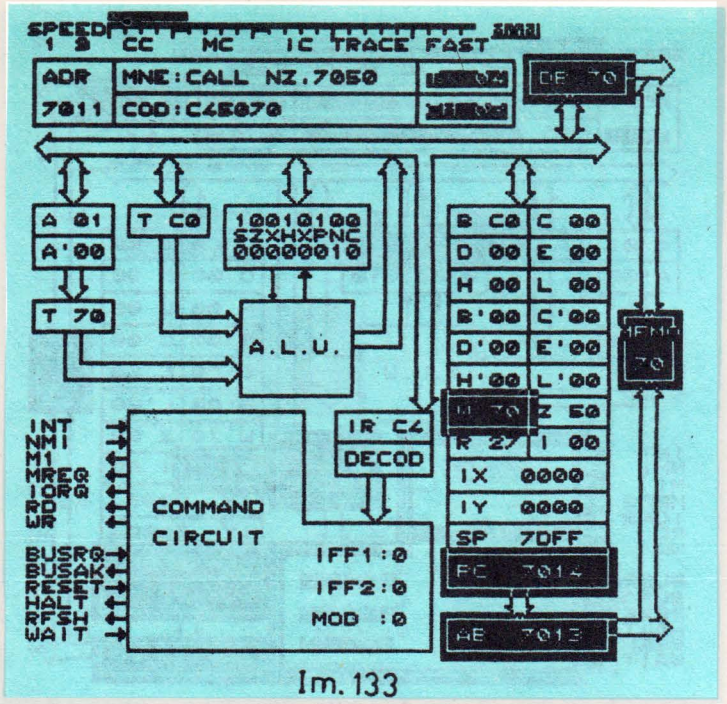
Dacă o secvență program poate fi folosită de mai multe ori pe parcursul elaborării unui program, atunci ea se va scrie o singură dată, sub forma unei subrutine. Astfel programul rezultat va fi mai scurt. Subrutina se distinge de restul secvențelor de cod prin


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
MNE:
COD:
DB C4
A 01
A'00
T C0
10010100
SZXHXPNC
00000010
T 70
A.L.U.
B C0
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 70
Z 0B
R 27
I 00
IX 0000
IY 0000
SP 7DFF
PC 7012
AB 7026
MEMO
C4
IR C4
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

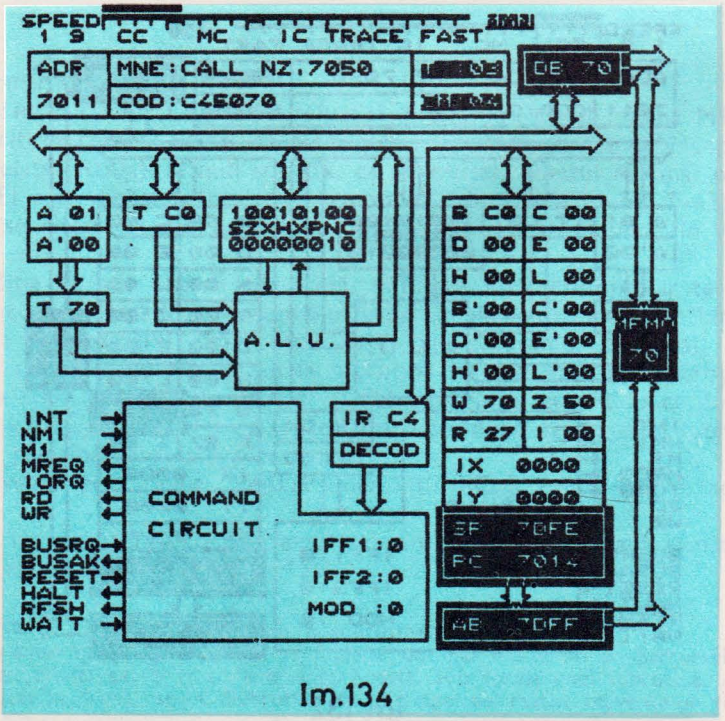
Im.131


SPEED
1 S
CC
MC
IC
TRACE
FAST
ADR
7011
MNE:CALL NZ,7050
COD:C45070
DB 50
A 01
A'00
T C0
10010100
SZXHXPNC
00000010
T 70
A.L.U.
B C0
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 70
Z 50
R 27
I 00
IX 0000
IY 0000
SP 7DFF
PC 7013
AB 7012
MEMO
50
IR C4
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

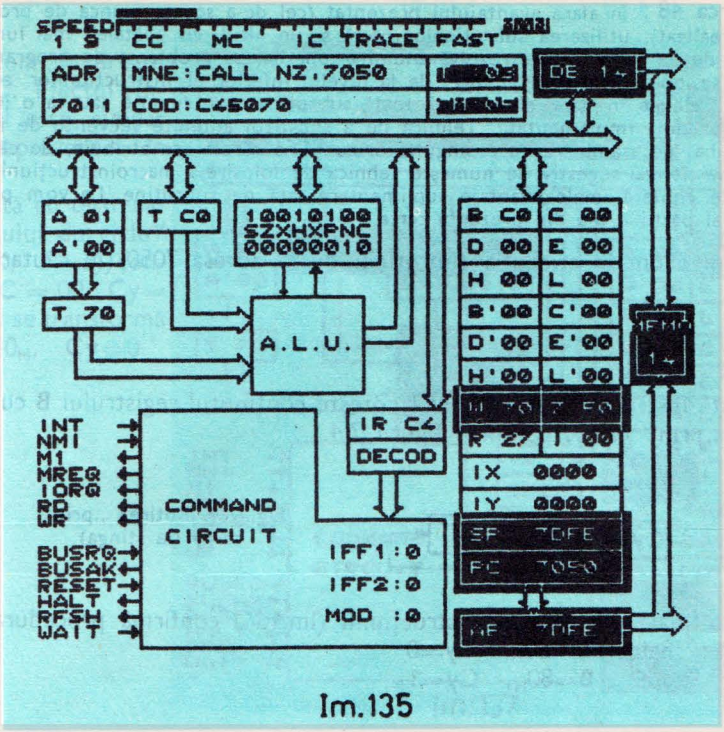
Im.132


SPEED
1 S CC MC IC TRACE FAST
ADR
7011
MNE:CALL NZ,7050
COD:C45070
DB 70
A 01
A'00
T C0
10010100
SZXHXPNC
00000010
T 70
A.L.U.
B C0 C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 70 Z 50
R 27 I 00
IX 0000
IY 0000
SP 7DFF
PC 7014
AB 7013
IR C4
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im. 133


SPEED
1 S CC MC IC TRACE FAST
ADR
7011
MNE:CALL NZ,7050
COD:C45070
DB 70
A 01
A'00
T C0
10010100
SZXHXPNC
00000010
T 70
A.L.U.
B C0 C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 70 Z 50
R 27 I 00
IX 0000
IY 0000
SP 7DFE
PC 7014
AB 7DFF
IR C4
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.134



Im.135

faptul că se termină cu o instrucțiune de revenire **(RET)** în programul apelant. Adresa de memorie la care începe secvența instrucțiunilor ce formează subrutina, se numește adresă de apel sau punct de intrare. Dorind să-i acordăm un plus de generalitate, programatorul poate realiza subrutina cu mai multe puncte de intrare și eventual mai multe puncte de revenire. În programul principal, execuția subrutinei se va prevedea incluzînd o instrucțiune de apel **(CALL)**, la întîlnirea căreia execuția programului principal se va abandona, trecîndu-se comanda în subrutină. După executarea ei, la întîlnirea unei instrucțiuni de revenire **(RET)**, execuția codului revine în programul apelant la instrucțiunea imediat următoare celei de apel subrutină. O subrutină poate apela o altă subrutină, care și ea la rîndul ei poate apela subrutine, etc.

Fenomenul se numește imbricare de subrutine. Datorită faptului că **Z80** memorează adresa de revenire în programul apelant pe stiva din memoria RAM, numărul nivelelor de imbricări a subrutinelor poate fi practic nelimitat. (Teoretic el este oricum limitat de dimensiunea fizică a memoriei **RAM**, 64 kocteți pentru microprocesorul **Z80**).

Instrucțiunile de apel și revenire din subrutine (clasa **CALL/RET**) pot fi condiționate sau necondiționate. Saltul la subrutină și înapoi se face totdeauna cu adrese absolute.

Condițiile de apel sau revenire se formulează la fel ca și în cazul instrucțiunilor de salt condiționat absolut.

CALL	NZ,	adresă	RET	NZ
CALL	Z,	adresă	RET	Z
CALL	NC,	adresă	RET	NC
CALL	C,	adresă	RET	C
CALL	PO,	adresă	RET	PO
CALL	PE,	adresă	RET	PE
CALL	P,	adresă	RET	P
CALL	M,	adresă	RET	M

Remarca 48 : În afara avantajului prezentat (cel de-a scurta munca de programare şi programul realizat), utilizarea **subrutinelor** oferă şi un altul, cu „bătaie" mai lungă. Creîndu-şi rutine de uz general, care rezolvă anumite funcţii (sau probleme de programare) programatorul se poate ridica în gîndire, de la nivelul inferior al instrucţiunilor cod maşină, la unul mai elevat în care el tratează toată subrutina ca o funcţie sau ca o instrucţiune microprocesor deja implementată. Tehnica de a substitui anumite secvenţe de instrucţiuni scrise în limbaj de asamblare cu o singură instrucţiune căreia i se atribuie, de către utilizator, un nume cît mai sugestiv, se numeşte tehnica de folosire a macroinstrucţiunilor. Macroinstrucţiunile conferă unele avantaje suplimentare faţă de subrutine. Le vom prezenta în capitolul 1 al părţii a 3-a din prezenta carte.

Trecem acum la execuţia subrutinei de la adresa 7050 (în căutarea bitului pierdut).

7050 RL B ;Im.136 (IC)

Această instrucţiune ROT/SHIFT roteşte conţinutul registrului **B** cu o poziţie la stînga, **„prin" carry**, conform figurii 7.4.

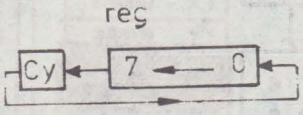
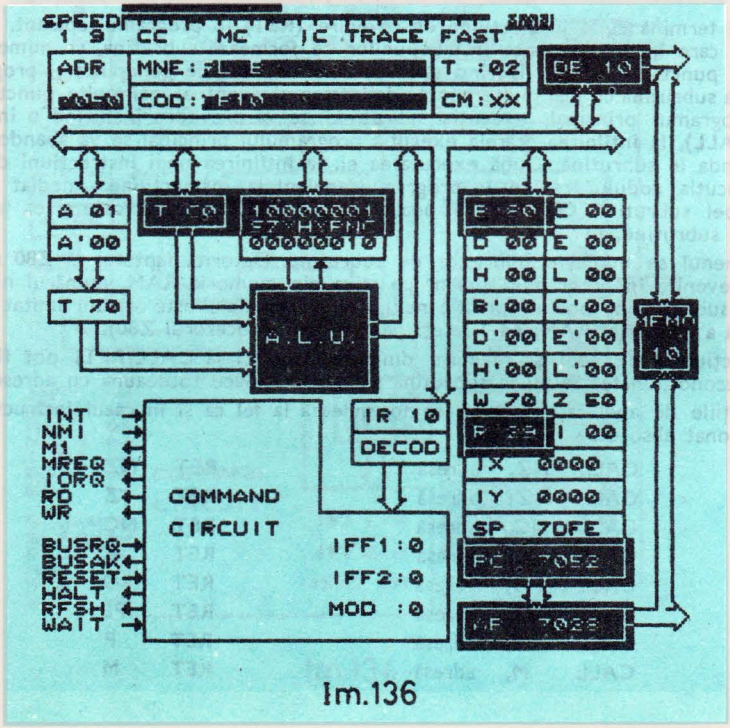


Fig. 7.4. Rotirea „prin" carry (la stînga)

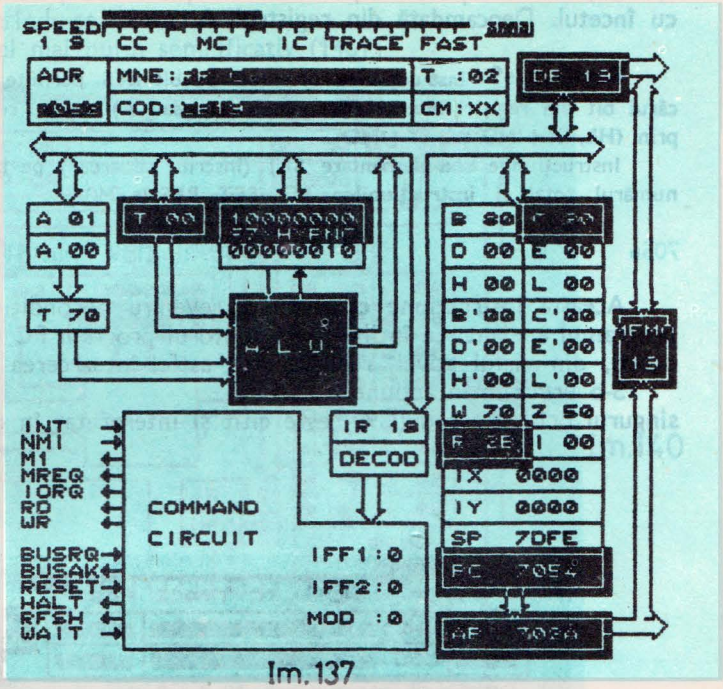
Starea finală a execuţiei instrucţiunii (Im.136) confirmă procedura ilustrată în fig. 7.4 : înainte $B=60_H$, $Cy=0$
după $B=80_H$, $Cy=1$



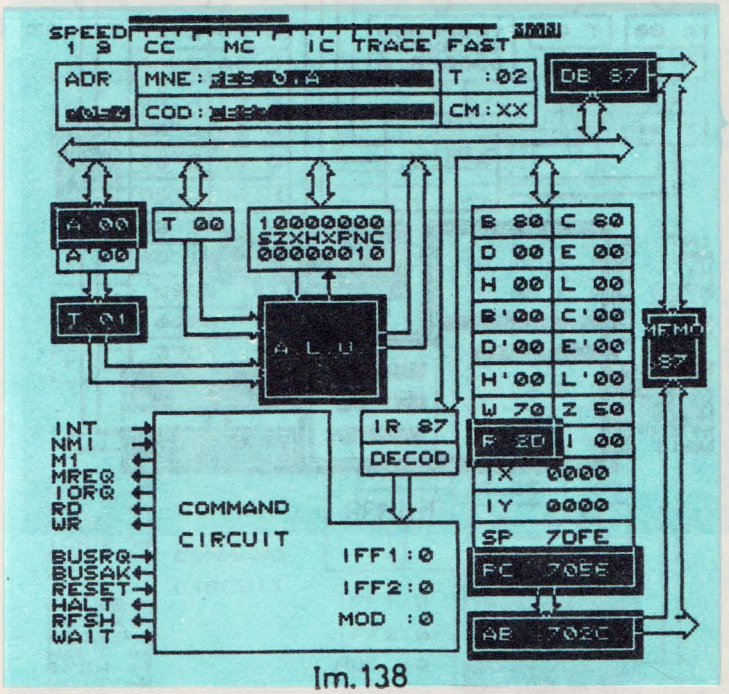
Im.136

Și acum „la dreapta", în C. Efectul instrucțiunii este similar cu cel reprezentat în fig. 7.1.

O dată cu terminarea „furtului de ciclu" (CM=XX, T=2) constatăm : C = 00 ; Cy= =1 inițial se transformă în C = 80_H, Cy=0 (Im.137)



Im.137



Im.138

Aici este începutul sfîrșitului. Tovarășul nostru de drum („1") dispare încetul cu încetul. Deocamdată din registrul A:

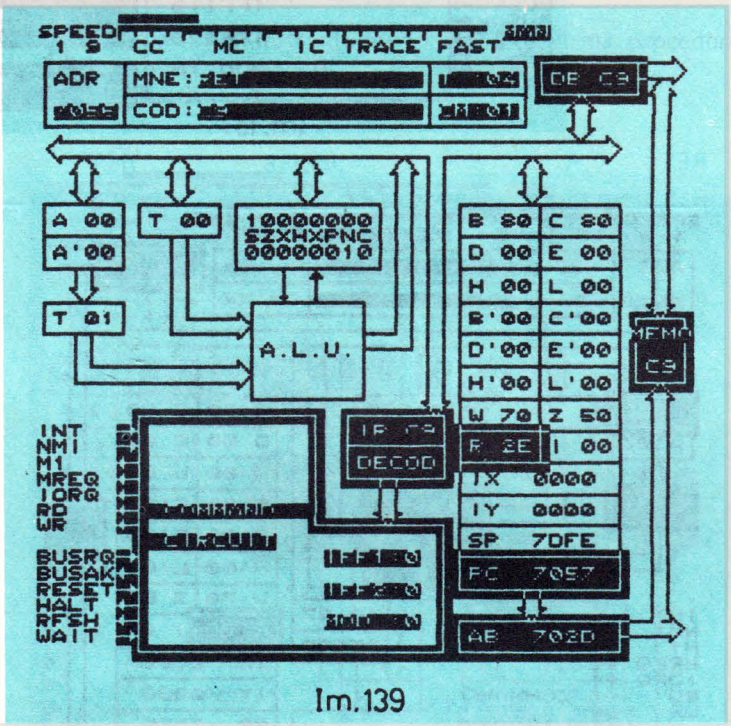
Remarca 49: Instrucțiunea orientată pe bit **RES** permite ștergerea selectivă a oricărui bit din regiștri interni (B, C, D, E, H, L sau A) și din celule de memorie adresate prin (HL, IX+IND sau IY+IND).

Instrucțiunile complementare **SET** (înscriu valoarea 1 pe bitul selectat) completează numărul total al instrucțiunilor **BIT, SET, RES** la 240.

7056 **RET** ;Im.139—Im.141 (MC)

Această instrucțiune cauzează o revenire necondiționată din subrutină în programul apelant. La întîlnirea ei contorul program PC este încărcat cu cei doi octeți, din vîrful stivei, realizîndu-se astfel întoarcerea la adresa corectă.

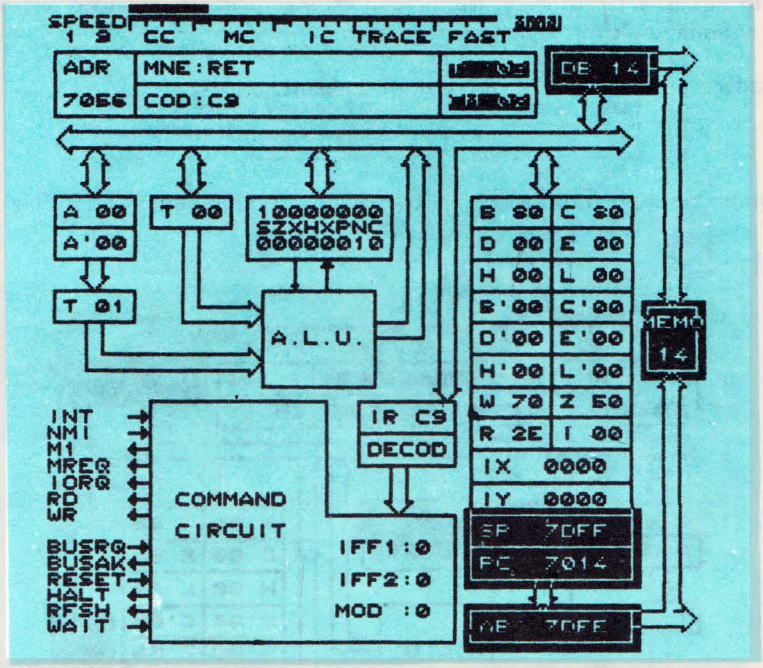
S-o urmărim în acțiune:
singurul octet de cod ($C9_H$) este citit și interpretat în ciclul fetch (Im.139).



Im.139

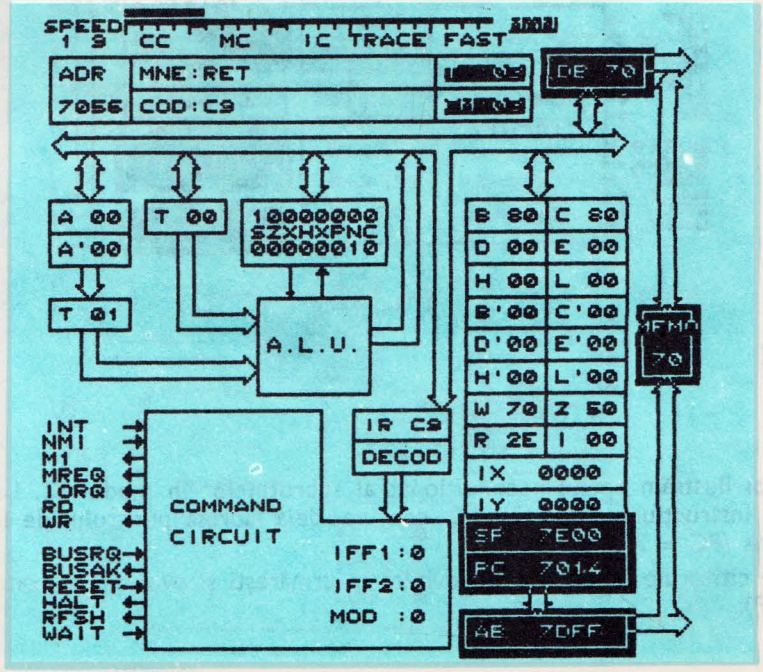
În ciclurile mașină M2 și M3 se efectuează două citiri din memorie, octeții din vîrful stivei transferîndu-se direct în PC.

Mai întîi octetul cel mai puțin semnificativ (14_H) :



Im.140

Apoi cel mai semnificativ (70_H) :



Im.141

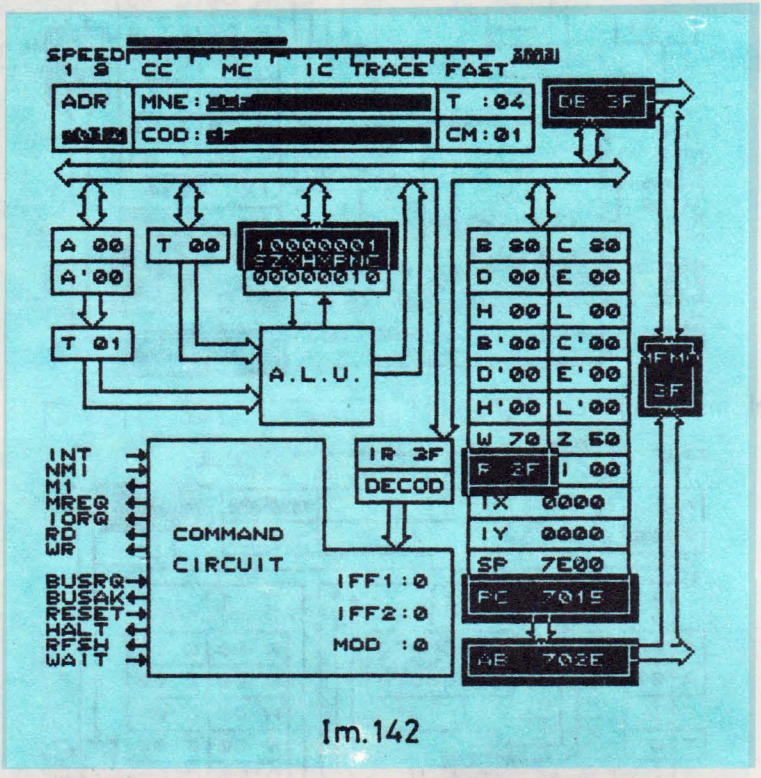
Remarca 50 : Instrucțiunile de revenire condiționată din subrutine (RET condiție) folosesc un ciclu fetch de 5 tacți procesor. Folosind revenirea necondiționată (RET) același ciclu va dura 4 tacți procesor.

Diferența își are originea în faptul că în cazul revenirii condiționate testarea condiției impuse necesită un ciclu suplimentar de tact.

Așadar, am revenit în programul apelant.

7014 CCF ;Im.142 (IC)

Complementăm indicatorul de transport (Complement Carry Flag) (Im.142).

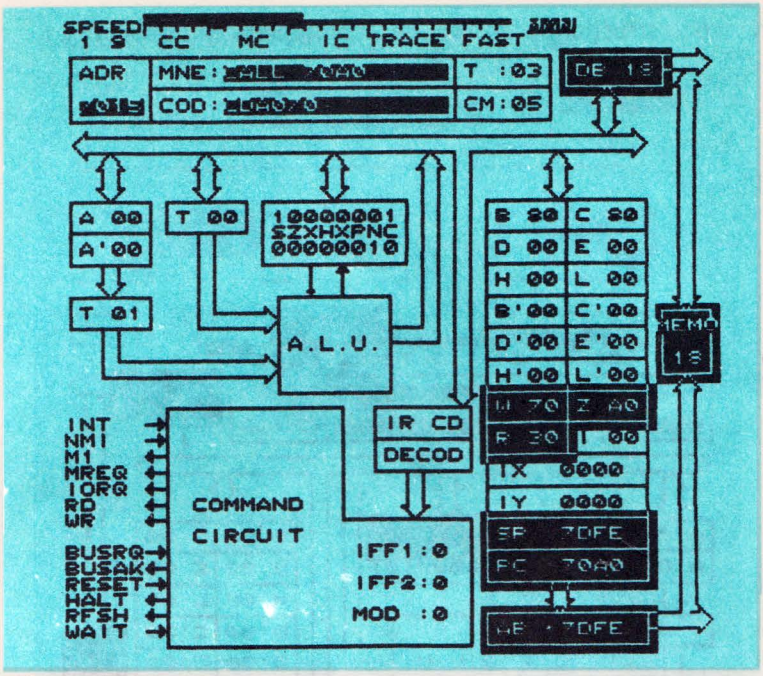


Im.142

7015 CALL 70A0 ;Im.143 (IC)

Apoi ilustrăm apelul necondiționat al subrutinelor în modul IC. La sfîrșitul ciclului instrucțiune (Im.143) PC conține deja adresa punctului de intrare în subrutină (PC = $70A0_H$).

(Pe tine cititorule binevoitor te invităm să urmărești și evoluția indicatorului de stivă SP).

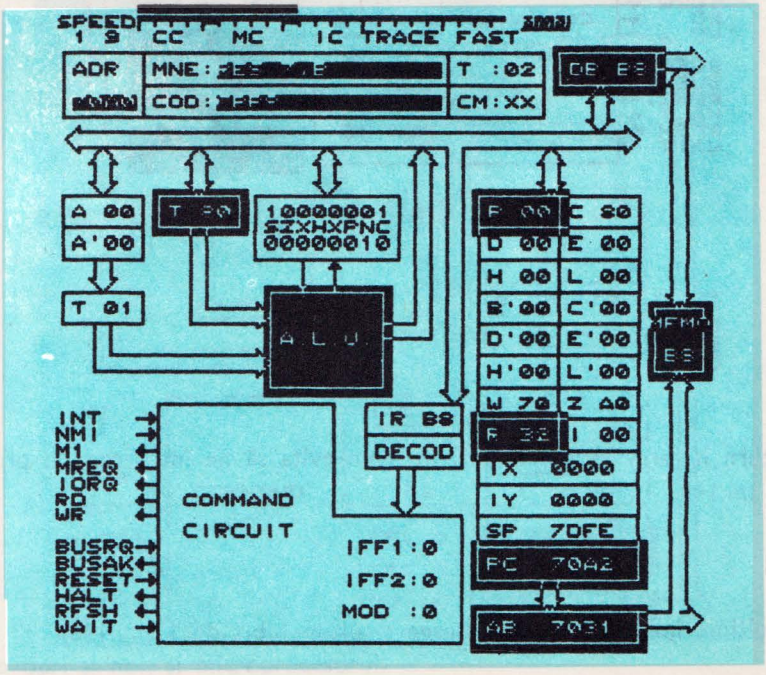


Im.143

Din nou în deplasare :

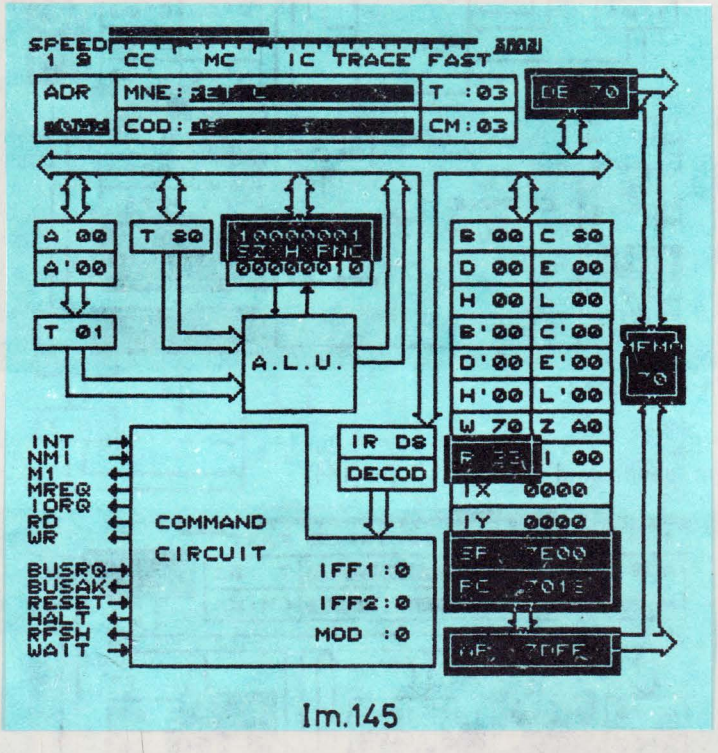
```
70A0      RES     7,B                ;Im.144          (IC)
```

Ştergem penultimul bit de „1" (Im.144)



Im.144

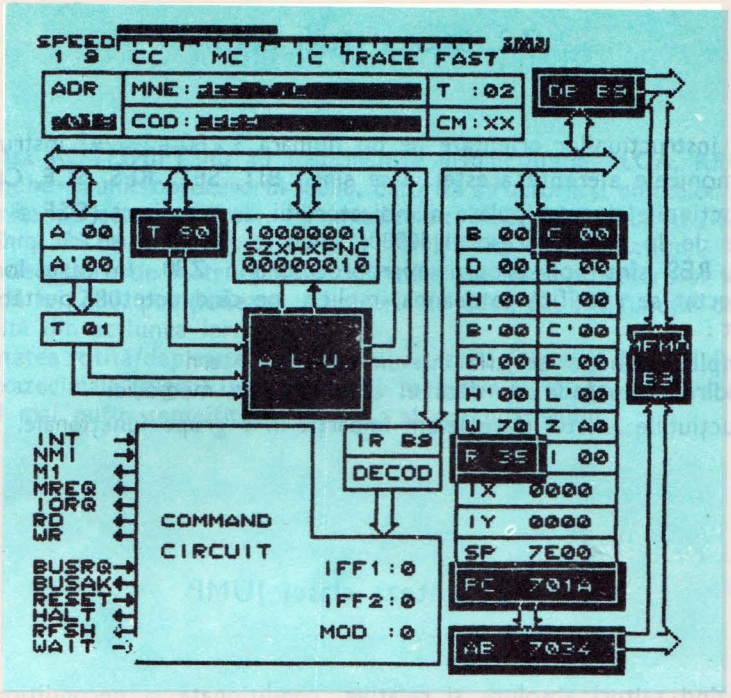
Să ilustrăm o instrucțiune de revenire condiționată din subrutină :



Im.145

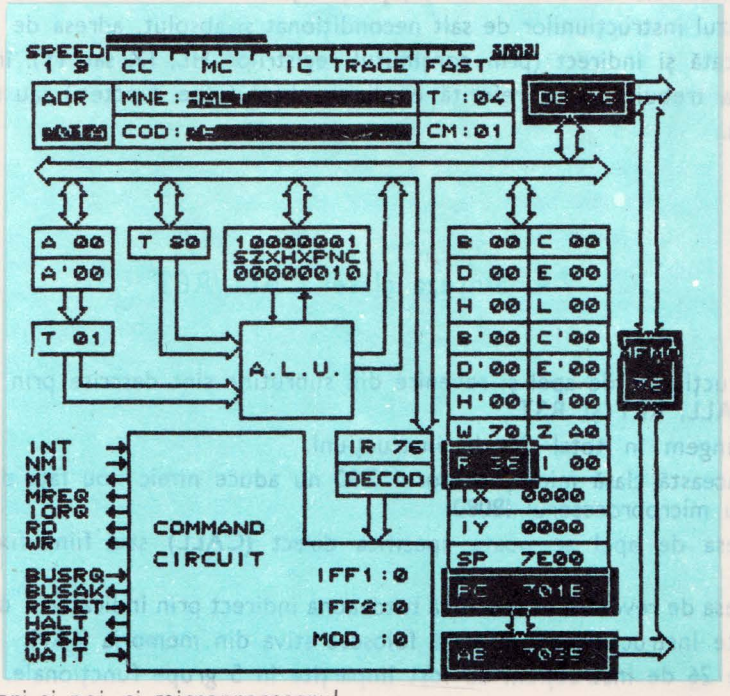
„Return if carry" ! Cy fiind 1, nu vom evita să ne întoarcem în programul apelant : Im.145.

„O ultimă îmbrățișare" : se șterge și ultimul bit de 1 (Im.146).



Im. 146

După care, executînd instrucțiunea **HALT**



Im. 147

ne vom opri și noi, și microprocesorul.

7.2. Sinteza clasei BITSR

Clasa instrucțiunilor orientate pe bit numără $3 \times 80 + 2 = 242$ instrucțiuni.

Mnemonicele aferente acestei clase sînt : **BIT, SET, RES, SCF, CCF.**

Instrucțiunile de manipulare a indicatorului de transport (**SCF** și **CCF**) sînt moștenite de la microprocesorul **I8080**, celelalte 240 descrise cu mnemonicele **BIT, SET, RES** sînt noi, proprii microprocesorului **Z80**. În cazul lor numărul bitului afectat se specifică totdeauna implicit, pe cînd octetul „purtător" poate fi adresat :

— implicit, cînd el se află într-un registru intern
— indirect sau indexat, dacă el este locat în memorie.

Instrucțiunile acestei clase le-am împărțit în 7 grupe funcționale.

7.3. Sinteza clasei JUMP

Incluzînd salturi absolute și relative, condiționate și necondiționate, clasa **JUMP** totalizează 18 instrucțiuni.

Mnemonicele aferente sînt : **JP, JR** și **DJNZ.**

În cazul instrucțiunilor de salt necondiționat și absolut, adresa de salt poate fi specificată și indirect (prin conținutul regiștrilor **HL, IX** sau **IY**), în celelalte cazuri, ea trebuie să fie referită explicit : absolut (pe 2 octeți) sau relativ (pe un octet).

7.4. Sinteza clasei CALL/RET

Instrucțiunile de apel și revenire din subrutine sînt descrise prin 3 mnemonice : **CALL, RET** și **RST.**

Distingem în total 26 de instrucțiuni.

La această clasă microprocesorul **Z80** nu aduce nimic nou față de predecesorul său microprocesorul **I8080.**

Adresa de apel se poate specifica direct **(CALL)** sau, fiind fixă, implicit **(RST).**

Adresa de revenire se specifică totdeauna indirect prin indicatorul de stivă **SP.**

Toate instrucțiunile din clasă folosesc stiva din memoria **RAM.**

Cele 26 de instrucțiuni au fost împărțite în 5 grupe funcționale.

7.5. Sinteza clasei ROT/SHIFT

În clasa **ROT/SHIFT** nu au fost incluse instrucțiunile **RLCA, RRCA, RLA, RRA,** comune cu microprocesorul **I8080,** datorită structurii și efectelor lor secundare diferite de cele ale clasei. Mnemonicele : **RLC, RRC, RL, RR, SLA, SRA, SRL, RLD,** și **RRD** reunesc în clasa **ROT/SHIFT** 72 de instrucțiuni.

Spre deosebire de instrucțiunile de rotire ale microprocesorului **I8080,** cele noi, proprii lui **Z80,** poziționează indicatorii de condiție. (În funcție de numărul care rezultă din acțiunea lor.)

Cantitatea rotită/deplasată este octetul sau cifra (digitul). **RLD** și **RRD** rotesc cifrele hexazecimale (digiți) unei locații de memorie la stînga sau la dreapta prin digitul cel mai puțin semnificativ (b_0—b_3) al acumulatorului.

8

INSTRUCȚIUNI DE INTRARE/IEȘIRE

Prin esența lor, instrucțiunile de intrare-ieșire sînt instrucțiuni de transfer. Datorită faptului că, pentru execuția lor se folosesc tehnici speciale, precum și datorită faptului că ele „sparg" granițele sistemului propriu-zis, comunicînd cu lumea externă, ele sînt tratate ca instrucțiuni aparte. Dacă numărul cererilor de intrare/ieșire este scăzut, atunci și frecvența de apariție probabilistică este redusă, ele putînd fi rezolvate eficace și în timp util de însăși „creierul" sistemului (unitatea centrală). Dacă în schimb numărul lor depășește un anumit nivel, ele pot ridica probleme, fie datorită faptului că nu pot fi servite la timp, fie că obligă unitatea centrală să-și „piardă" prea mult din timpul ei prețios, pentru rezolvarea acestor probleme doar aparent minore.

Pe cînd în al doilea caz, — debarasarea unității centrale de probleme de I/E — este doar recomandabilă, „statul deoparte" devine obligatoriu atunci cînd ea nu poate realiza în timp util (real) cererile de I/E, pierzînd astfel nu numai informații prețioase, ci riscînd să piardă total contactul cu lumea externă. În asemenea situații poate apare blocarea sistemului, fenomen care se califică drept eroare fatală. Dacă blocajele unui sistem sînt frecvente, atunci el este pe bună dreptate apostrofat, spunîndu-se : „nu-i bun la nimic !" Pentru a evita această situație au fost elaborate subunități funcționale specializate pe probleme de intrare/ieșire, care pot rezolva favorabil toate problemele de intrare/ieșire în mod autonom, fără a implica masiv unitatea centrală a sistemului. În sistemele mari dotate cu astfel de facilități — numite canale de intrare/ieșire — unitatea centrală lucrează pe problemele sale și în timpul transformărilor de date efectuate de către canale, pe cînd în cele mici ea este de obicei suspendată în acest răstimp, unitatea specializată de intrare/ieșire preluînd comanda în sistem.

Modul în care un sistem își rezolvă problemele de intrare/ieșire, poate fi un criteriu calitativ determinant.

Fiind vorba de instrucțiuni de I/E ați realizat probabil deja, faptul că sistemul nostru rezolvă problemele „cu mînă proprie" adică cu participarea directă a unității centrale.

8.1. Ciclurile mașină de intrare și ieșire

Pe lîngă ciclurile mașină de bază **(FETCH, READ, WRITE)** instrucțiunile de intrare/ieșire efectuează și anumite cicluri mașină specifice, numite cicluri de intrare/ieșire. Pe cînd ciclurile de bază au ca trăsătură comună activarea semnalului procesor $\overline{\text{MREQ}}$, ciclurile de intrare/ieșire sînt concretizate de activarea semnalului $\overline{\text{IORQ}}$ (Input Output ReQuest). Pe parcursul lor, are loc transferul efectiv al unui octet între microprocesor și dispozitivul de intrare/ieșire selectat. Ele pot fi cicluri de intrare **(IN)**, sau de ieșire **(OUT)**. Microprocesorul **Z80** semnalează tipul ciclului mașină de I/E în curs, prin activarea (simultan cu $\overline{\text{IORQ}}$) a unuia din semnalele de comandă $\overline{\text{RD}}$ (Read) sau $\overline{\text{WR}}$ (Write).

Fiind deja familiarizați cu diagramele de timp, prezentăm ciclurile I/E în ordinea inversă față de cea folosită la ciclurile mașină de bază. Mai întîi realitatea (diagramele de timp) și doar apoi poveștile (aventură **VISIBLE=Z80**).

Aidoma tuturor ciclurilor mașină ale microprocesorului **Z80** și cele de intrare/ieșire încep pe frontul crescător al semnalului de tact Ø.

Ciclul IN

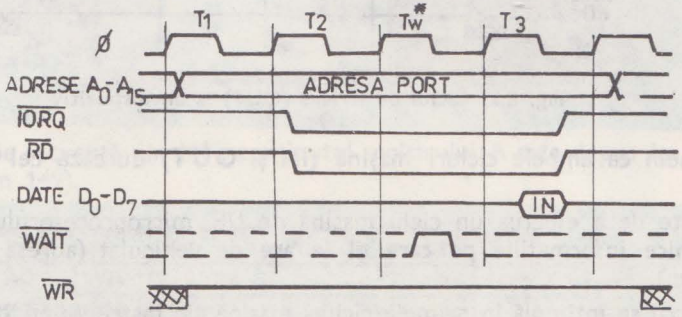


Fig. 8.1. Ciclul de citire (IN) de la un dispozitiv

Așa cum reiese din fig. 8.1 ciclul **IN** începe cu activarea liniilor de adresă : pe liniile mai puțin semnificative **A0 — A7** apare adresa portului de citit, adresă care provine dintr-unul din regiștri interni ai microprocesorului. La începutul celui de-al doilea ciclu de tact **(T2)** se activează semnalele $\overline{\text{IORQ}}$ și $\overline{\text{RD}}$ specificînd tipul ciclului mașină în curs : **IN**.

Spre deosebire de ciclurile mașină de bază (cu acces la memorie) unde semnalele de comandă $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ se activează pe frontul scăzător al primului tact din ciclu **(T1)**, la ciclurile mașină de I/E, semnalele de comandă se generează cu o întîrziere de jumătate de tact procesor. Pentru a permite dispozitivelor de intrare/ieșire să se selecteze în timp util, microprocesorul inserează un ciclu de tact suplimentar **Tw** (vom vedea în cap. 9 ce înseamnă **w** — Wait).

Considerînd că în 2,5 cicluri de tact dispozitivul de intrare apelat a reușit să-și plaseze datele pe magistrala de date **(D0—D7)**, pe frontul scăzător al celui de-al patrulea ciclu de tact **(T3)** microprocesorul citește octetul de pe liniile de date, dezactivînd apoi semnalele de comandă $\overline{\text{RD}}$ și $\overline{\text{IORQ}}$. Pe tot parcursul ciclului **IN** semnalul $\overline{\text{WR}}$ este inactiv.

Ciclul OUT

Mai întîi se generează adresa portului de ieşire (vezi fig. 8.2), pe **A0—A7.** După o jumătate de tact, procesorul plasează pe magistrala de date octetul de emis. Pe frontul crescător al ciclului de tact **T2** se activează semnalele de comandă $\overline{\text{IORQ}}$ şi $\overline{\text{WR}}$.

Din aceleaşi motive, amintite la **IN,** microprocesorul inserează un ciclu de tact suplimentar, urmînd să dezactiveze mai întîi semnalele de comandă (front scăzător în **T3**) iar apoi liniile de date.

Semnalul $\overline{\text{RD}}$ este bineînţeles inactiv pe toată perioada ciclului **OUT.**

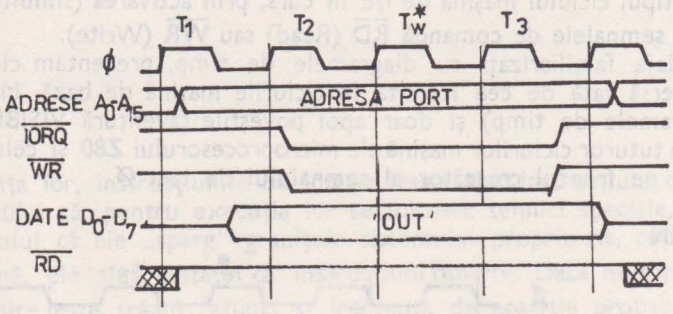


Fig. 8.2. Ciclul de scriere (OUT) la un dispozitiv

Reţinem că ambele cicluri maşină **(IN** şi **OUT)** durează cel puţin 4 ciclu r de tact.

Înainte de a efectua un ciclu maşină de I/E, microprocesorului trebuie să l se comunice informaţiile pe care el le are de vehiculat (adresă port, date de emis, etc.).

Acest lucru se întîmplă în primele cicluri maşină ale instrucţiunii de I/E, aşa cum vom vedea în aventura următoare.

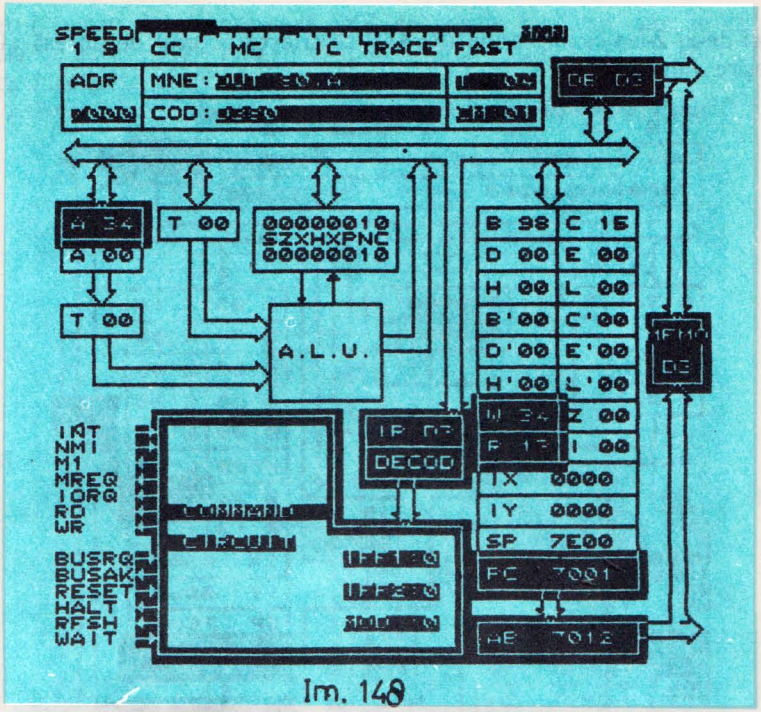
8.2. Aventura a 7-a: intrare/ieşire

De astă dată drumul nostru va fi scurt :

```
7000    OUT     (80),A      ;Im.148— Im.155   (MC,CC)
7002    IN      D, (C)      ;Im.156— Im.158   (MC)
```

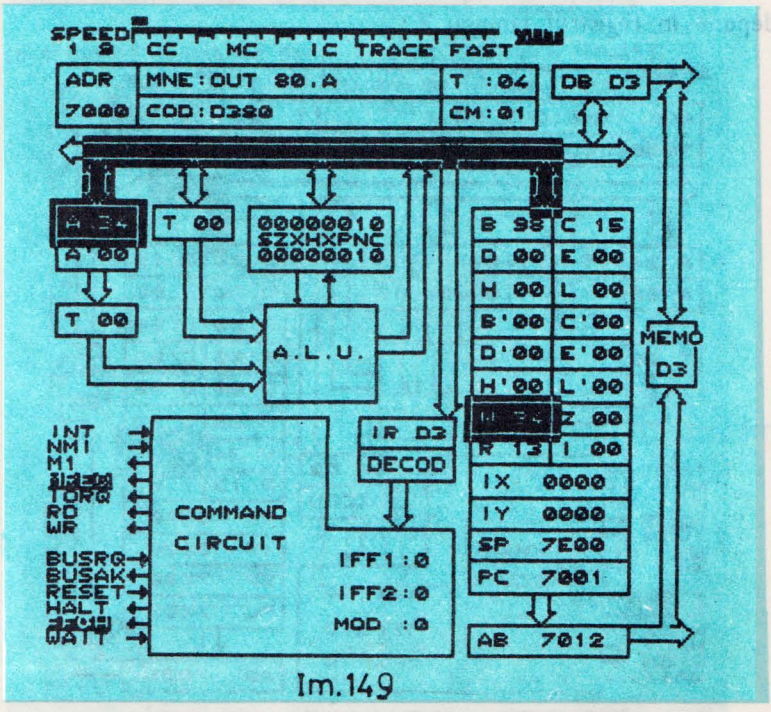
Instrucţiunea de ieşire cu adresare directă **OUT (80),A** transferă conţinutul registrului **A** în dispozitivul (portul) de ieşire cu adresa **80H.** Ea ocupă în memorie 2 octeţi. Primul octet este codul instrucţiunii, iar cel de-al 2-lea adresa portului de selectat.

În primul ciclu maşină (Im.148) se citeşte şi se interpretează codul instrucţiunii : $D3_H$.



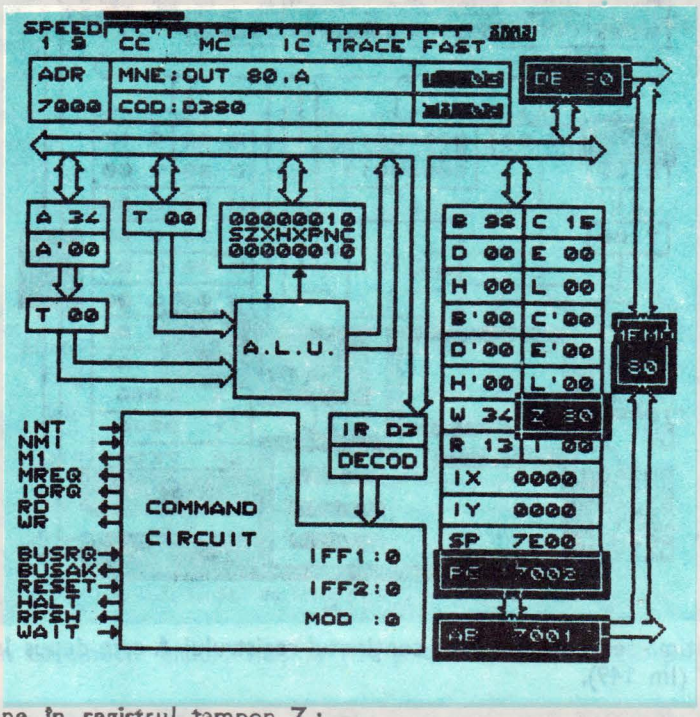
Im. 148

Reţinem ultima secvenţă din **M1** : conţinutul registrului **A** este depus în registrul tampon **W** (Im 149).



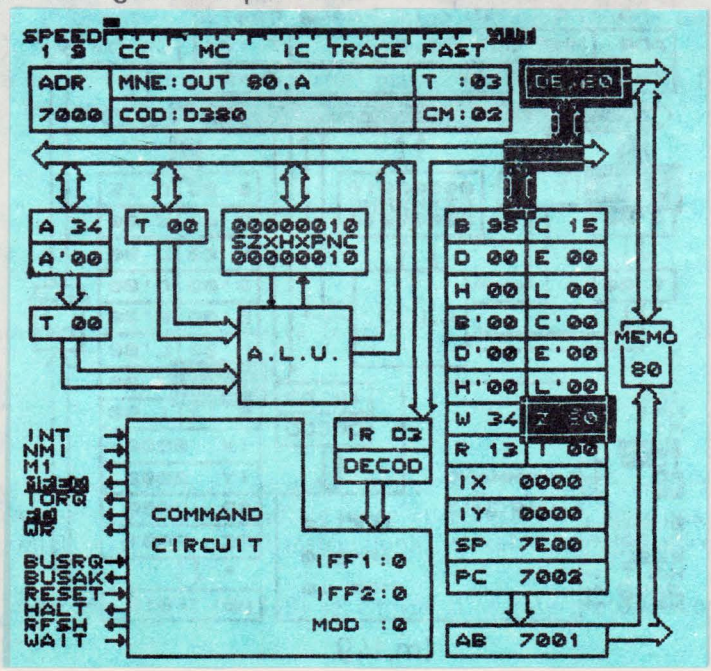
Im.149

În cel de-al 2-lea ciclu mașină (READ) se citește octetul de adresă al portului de ieșire (Im.150),



Im. 150

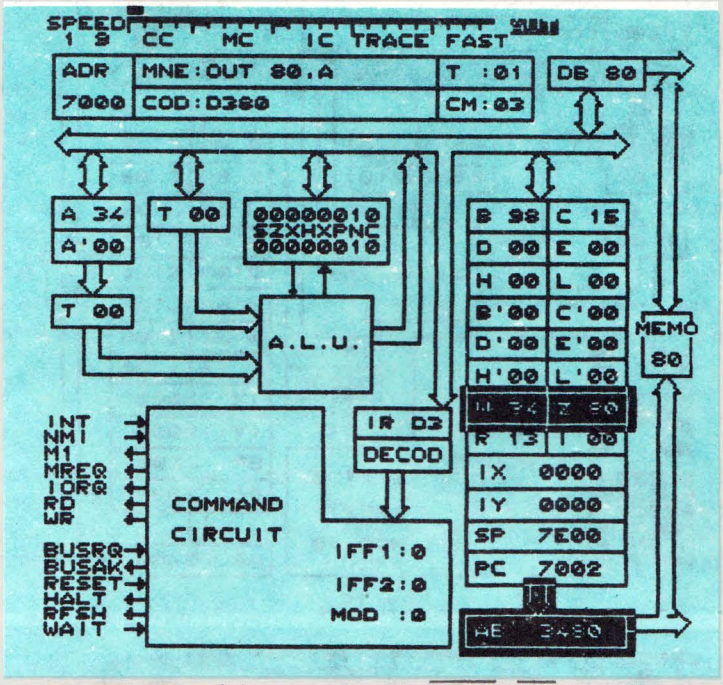
și se depune în registrul tampon Z:



Im. 151

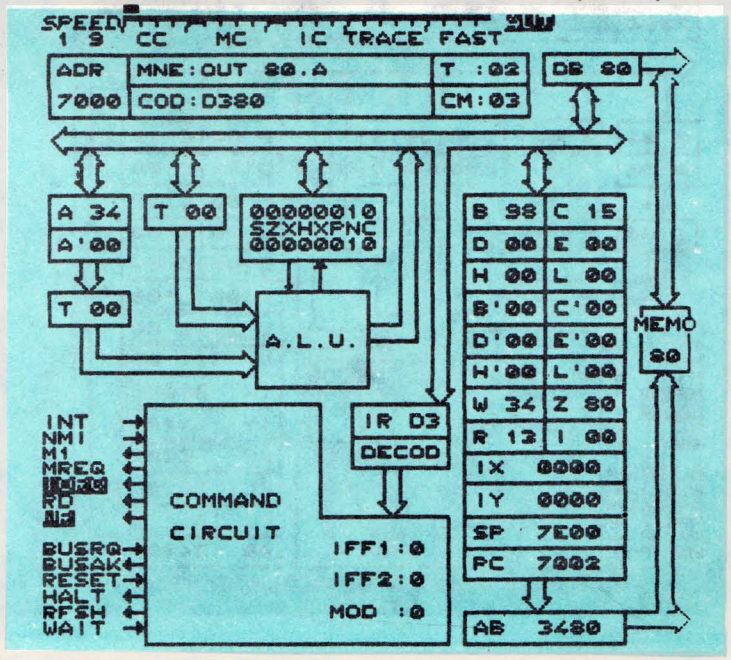
Conținutul registrului transparent WZ constituie cuvîntul care se va plasa pe magistrala de adrese A0—A15 în timpul ciclului mașină **OUT**. Îl vom „filma" pe acesta cu încetinitorul (CC).

Se activează liniile de adrese, plasînd pe ele conținutul amintit (Im.152)



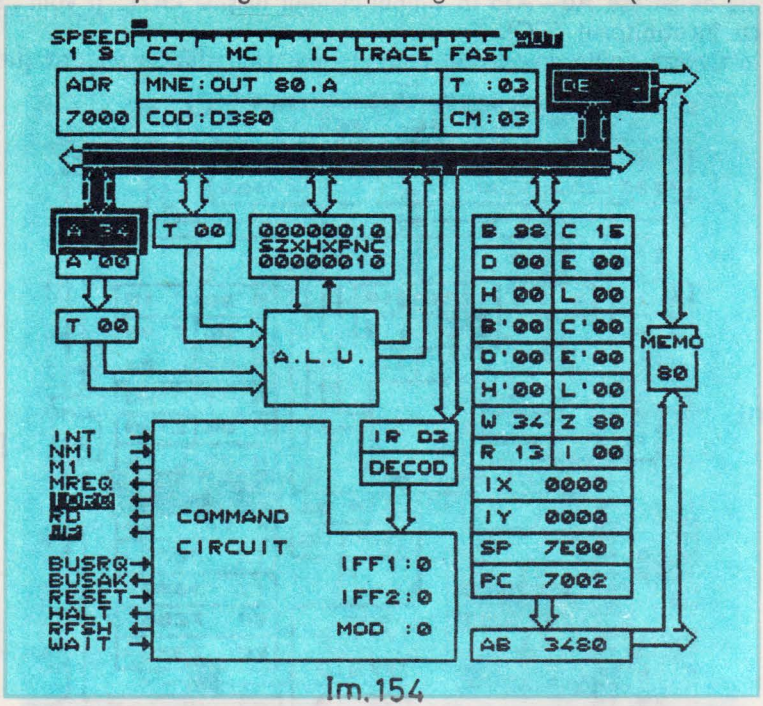
Im.152

iar apoi se activează semnalele de comandă $\overline{IORQ}$, $\overline{WR}$ (Im.153).

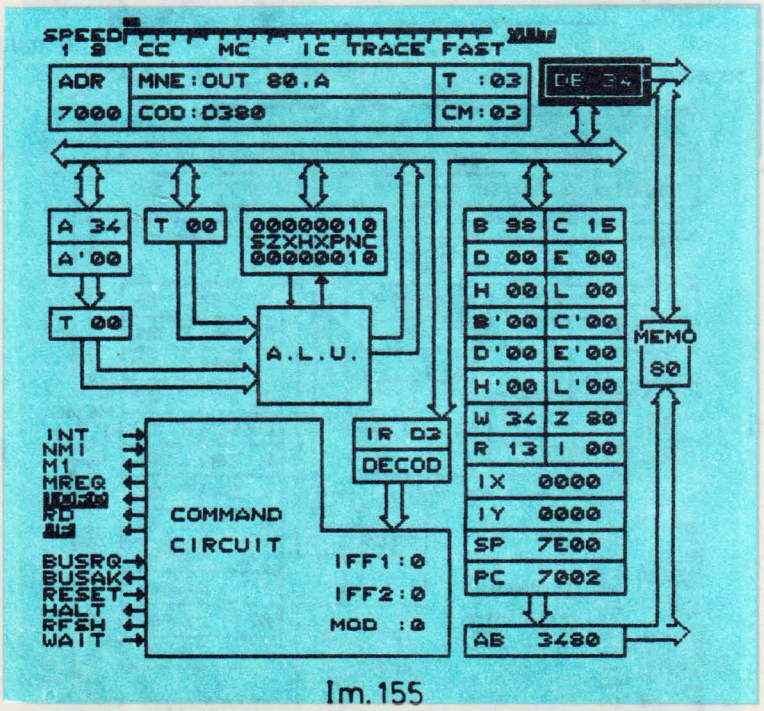


Im.153

În fine, se emite conținutul registrului A pe magistrala de date (Im.154 și Im.155).



Im.154



Im.155

Remarca 51 : Cei care au citit atent paragraful precedent şi au urmărit în aceeaşi manieră imaginile de sus, au remarcat probabil două contradicţii.

1. În imaginile Im.153 şi Im.154 semnalele de comandă se activează înainte de emiterea datelor, invers decît se întîmplă în realitate. Asta este, trebuie să recunoaştem, o eroare a **VISIBLE—Z80**.

2. Ciclul **OUT** s-a terminat în 3 cicluri de tacţi şi nu în 4, cum afirmăm pe baza diagramelor de timp. În **VISIBLE—Z80** am neglijat ciclul de tact suplimentar **Tw**.

Remarca 52 : Spre deosebire de microprocesorul **I8080**, la care adresa portului de ieşire se repetă şi pe liniile **A8—A15**, în cazul acestei instrucţiuni **Z80** emite pe liniile superioare conţinutul registrului **A**.

Remarca 53 : Pentru a selecta un anumit port de ieşire (intrare), proiectantul (hardware) va trebui să decodifice folosind circuite logice combinaţionale, sau alte tehnici, liniile de adresă **A0—A7**.

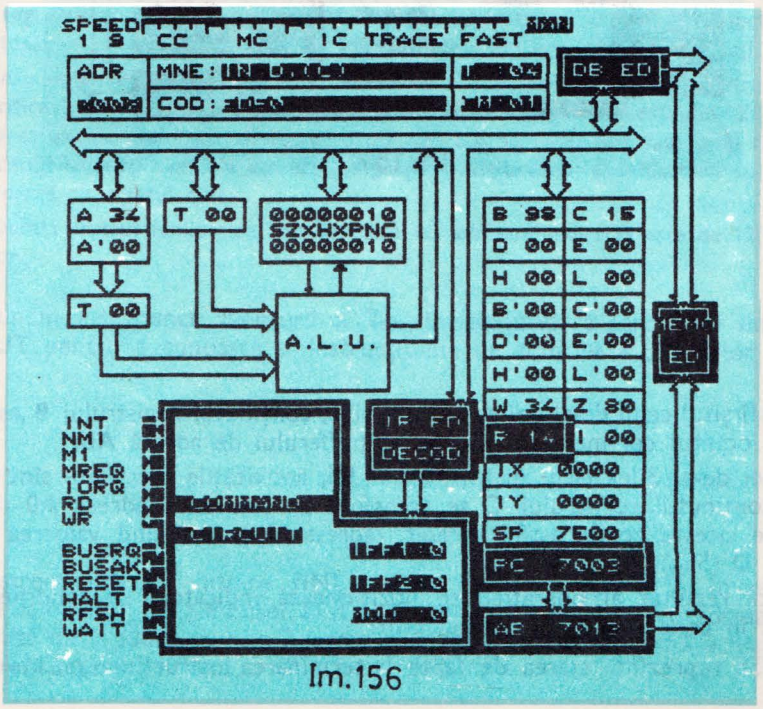
Remarca 54 : Decodificînd toată magistrala **A0—A15** în ciclurile **IN** şi **OUT**, avem posibilitatea de a selecta 2^{16}=65536 dispozitive de intrare şi tot atîtea de ieşire. Chiar dacă numărul este total iluzoric, notăm acest plus de performanţă a microprocesorului **Z80** faţă de **080**.

```
7002    IN      D,(C)          ;Im.156—Im.158  (MC)
```

Instrucţiunea de faţă citeşte conţinutul portului de intrare a cărui adresă este cuprinsă în registrul **C**, şi îl transferă în registrul **D** al microprocesorului. Instrucţiunea ocupă 2 octeţi în memorie, ambii fiind octeţi de cod.

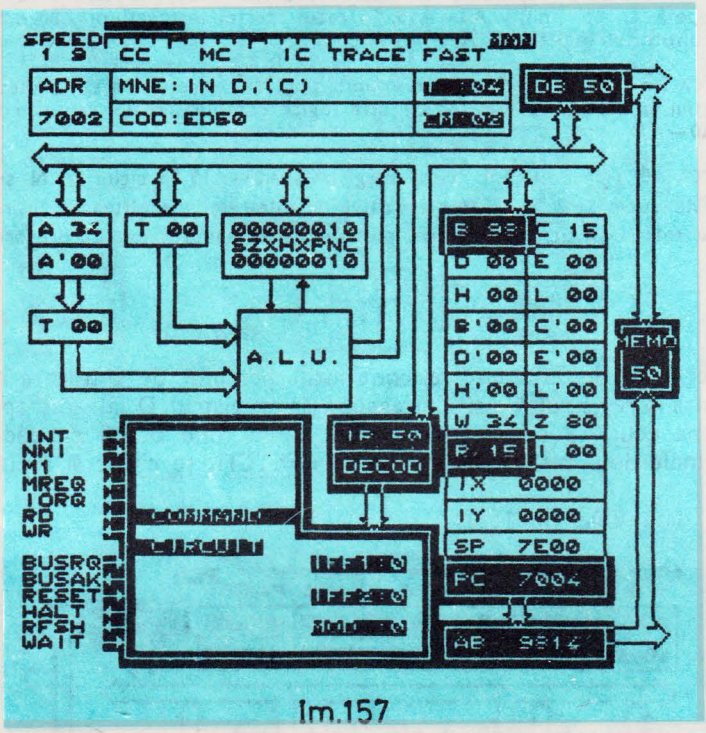
În primele două cicluri maşină, ambele **FETCH**, se citesc şi se interpretează codurile.

Prima dată ED_H (Im.156):



Im.156

iar apoi $E0_H$:



Im.157

La sfîrșitul celui de-al doilea ciclu mașină, conținutul registrului **B** este transferat în octetul cel mai semnificativ al bufferului de adresă **AB**.

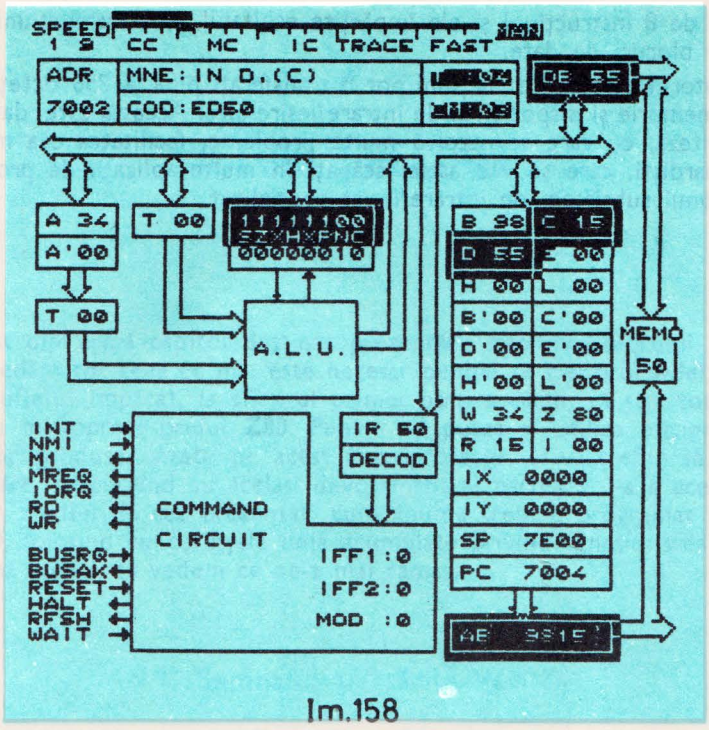
În cel de-al 3-lea ciclu mașină, ciclul **IN**, activitățile efectuate sînt :

— conținutul registrului **C** se transferă pe liniile de adresă **A0—A7**

— se citește portul astfel adresat, (adresa 15_H), încărcînd valoarea citită în registrul **D** (55_H)

— ca rezultat al operației, se poziționează indicatorii de condiție a registrului **F**, în concordanță cu octetul citit.

Im.158 reprezintă starea de fapte la terminarea instrucțiunii : CM=3, T=4.



Im.158

Remarca 55 : Instrucțiunile **IN** și **OUT** pot folosi adresarea directă sau indirectă, „via" registrul **C**. Cele cu adresare directă sînt preluate de la **Intel**, ele efectuînd transferul totdeauna prin registrul **A**. Necesarul lor de timp este 11 (4, 3, 4) tacți procesor.

Instrucțiunile **IN, OUT** cu adresare indirectă folosesc conținutul registrului **C** pentru a specifica adresa perifericului dorit. Ele sînt proprii microprocesorului **Z80**, permițînd efectuarea transferului prin oricare registru intern : **B, C, D, E, H, L** sau **A**. În fine, așa cum am văzut, instrucțiunile **IN** din această grupă poziționează și indicatorii de condiție în concordanță cu octetul citit.

Durata instrucțiunilor de intrare/ieșire cu adresare indirectă este de 12(4, 4, 4) tact procesor.

Punem aici punct aventurii a 7-a, invitîndu-vă să parcurgeți sinteza clasei **IN/OUT** pentru a cunoaște și instrucțiunile de transfer a blocurilor de date.

8.3. Sinteza clasei IN/OUT

Instrucțiunile de intrare **(IN)** și cele de ieșire **(OUT)** își împart frățește cele 24 de instrucțiuni ale clasei. Distingem cîte o instrucțiune cu adresare directă, moștenită de la I8080, și cîte 7 instrucțiuni cu adresare indirectă a dispozitivului de intrare/ieșire.

Restul de 8 instrucțiuni și ele împărțite echitabil, sînt instrucțiuni de transfer a unor blocuri de date.

Cu ajutorul acestor instrucțiuni pot fi transferați pînă la 256 octeți, dintr-un foc, între memorie și dispozitivul de intrare/ieșire ales. Aceasta este, datorită sporului de viteză, cu care se rezolvă multe probleme, facilitatea cea mai gustată de către hardiști, care se văd astfel scăpați în multe aplicații de proiectarea și realizarea unui subsistem de intrare/ieșire specializat.

9

CICLURI SPECIALE

În acest din urmă capitol destinat prezentării microprocesorului **Z80**, vom încerca să redăm tot ceea ce mai este necesar pentru ca cititorul fidel, să poată afirma cu sufletul împăcat, la sfîrșitul primei părți a cărții, că știe totul despre funcționarea microprocesorului **Z80**. Pentru a-l putea și utiliza eficient, în elaborarea unor produse bazate pe acest microprocesor, el va trebui să mai aibă puțină răbdare, parcurgînd cu același devotament și partea a 2-a a acestei cărți, învățînd din studiul de caz prezentat, cum anume trebuie programat microprocesorul **Z80**, folosind cunoștințele deja acumulate privind funcționarea lui.

Dar pînă atunci, să vedem ce ne-a mai rămas.

9.1. Semnalul și stările WAIT

Aminteam în cap. 3., la prezentarea semnalelor de stare, că este de datoria unui conducător ca, după emiterea unei porunci, să arunce o privire asupra subordonaților lui, pentru a se convinge că cei chemați să ducă la îndeplinire ordinul lui, sînt într-adevăr capabili să o facă în timp util. Știind că aceștia îi sînt parteneri, el n-are încotro, va trebui să mai aștepte în cazul în care constată că răspunsul va întîrzia. Transpunînd fenomenul în lumea microprocesoarelor, semnalul $\overline{\text{WAIT}}$ (așteaptă) va fi acela pe care unitatea de comandă a microprocesorului îl va interoga după emiterea fiecărei comenzi pentru a ști dacă ea a fost îndeplinită cu succes, sau va trebui să mai aștepte. Dacă elementele componente ale sistemului sînt mai rapide decît o cere frecvența de tact a microprocesorului, atunci proiectantul poate uita de existența semnalului $\overline{\text{WAIT}}$ (legîndu-l printr-o rezistență de aproximativ 1 kΩ la +5V). În caz contrar, elementele mai lente vor trebui să detecteze faptul că sînt apelate, și să strige cu glas tare (activînd semnalul $\overline{\text{WAIT}}$), cerînd „stăpînului" să mai aștepte. Detectînd o cerere de așteptare ($\overline{\text{WAIT}}$), microprocesorul va amîna finalizarea ciclului mașină în curs, inserînd cîte un ciclu de tact de așteptare și verificînd de fiecare dată starea semnalului $\overline{\text{WAIT}}$, pînă la dezactivarea acestuia. Astfel durata oricărui ciclu mașină care face apel la resursele hardware externe procesorului, se poate prelungi oricît de mult — cu un număr întreg de cicluri de tact —, chiar și la infinit.

Trecînd peste această situație limită, absurdă în felul ei, constatăm că existența semnalului $\overline{\text{WAIT}}$ permite sincronizarea procesorului oricît de rapid ar fi, cu elemente hardware mai lente, existente în sistem, creîndu-se astfel condițiile

pentru conviețuirea pașnică a unei diversități mari de componente electronice, începînd cu memorii de ferită avînd un timp de acces de ordinul microsecundelor și terminînd cu memorii semiconductoare bipolare, a căror timp de acces este de ordinul zecilor de nanosecunde.

Cum orice activitate a microprocesorului începe și se termină pe frontul crescător al semnalului de tact Ø, este lesne de înțeles că interogarea semnalului $\overline{\text{WAIT}}$ se va face pe frontul descrescător al aceluiași semnal. Astfel unitatea de comandă a microprocesorului va avea timp pentru a insera, în caz de nevoie, o stare de așteptare (WAIT), stare care de asemenea începe cu frontul crescător al lui Ø.

Vom ilustra, folosind diagrame de timp, efectul activării semnalului $\overline{\text{WAIT}}$ asupra fiecăruia din ciclurile mașină studiate pînă în prezent.

Ciclul FETCH cu stări de așteptare.

Semnalul $\overline{\text{WAIT}}$ este eșantionat pe frontul descrescător al celui de-al 2-lea ciclu de tact (T2). Dacă el este activ în acel moment, atunci se va insera o stare de WAIT, cu durata egală cu perioada ciclului de tact. În mijlocul stării WAIT (frontul descrescător al tactului în Tw) microprocesorul verifică din nou linia $\overline{\text{WAIT}}$.

Dacă ea este activă și în acest moment, atunci se inserează o nouă stare de WAIT. După dispariția cererii de așteptare, execuția ciclului FETCH continuă cu ciclul de tact T3. (vezi figura 9.1).

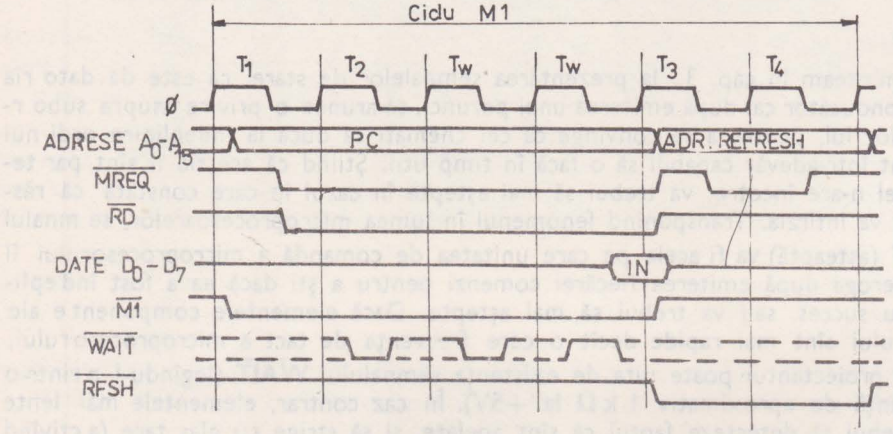


Fig. 9.1. Ciclul M1 (FETCH) cu stări WAIT

Reținem că în afara momentelor de timp specificate, starea semnalului $\overline{\text{WAIT}}$ este neinteresantă.

Ciclul READ cu stări WAIT.

Prelucrarea eventualelor cereri de așteptare se face în aceeași manieră ca și în ciclul FETCH eșantionarea semnalului $\overline{\text{WAIT}}$ făcîndu-se pe frontul descrescător al semnalului de tact în ciclul T2. (Vezi figura 9.2)

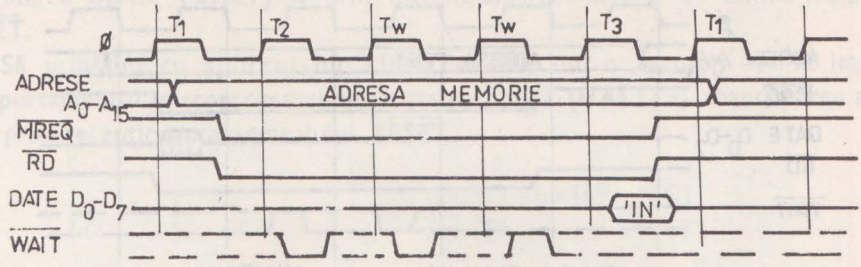


Fig. 9.2. Ciclul de citire memorie (READ) cu stări WAIT

Ciclul WRITE cu stări WAIT

Ca și în cazurile precedente, interogarea semnalului $\overline{WAIT}$ se face pe frontul scăzător al ciclului de tact **T2**. Fenomenologia prezentată la ciclul **FETCH** se repetă și la ciclul **WRITE** (Vezi figura 9.3).

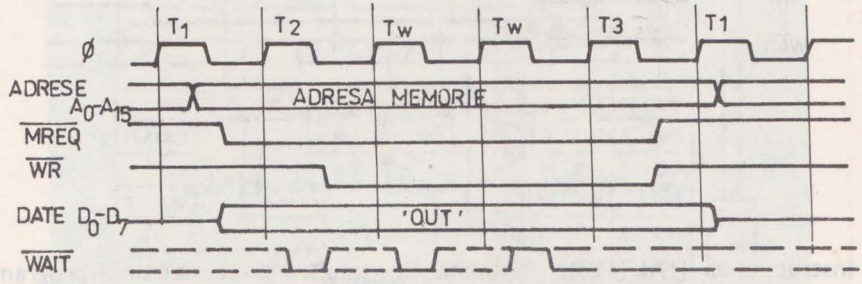


Fig. 9.3. Ciclul de scriere memorie (WRITE) cu stări WAIT

Ciclul IN cu stări WAIT

Datorită faptului că în ciclurile mașină de intrare/ieșire semnalele de comandă $\overline{IORQ}$ și $\overline{RD}$ respectiv $\overline{IORQ}$ și $\overline{WR}$ se generează de abia pe frontul crescător al ciclului de tact **T2**, verificarea cererii de așteptare pe frontul descrescător al aceluiași ciclu de tact ar fi prematură : s-ar putea ca dispozitivul de I/E apelat să nu fi avut timp suficient pentru a-și formula răspunsul (să activeze semnalul $\overline{WAIT}$). De aceea prima interogare se va face pe frontul scăzător al ciclului de așteptare **Tw*** pe care unitatea de comandă a microprocesorului **Z80** o inserează automat în fiecare ciclu mașină **IN** și **OUT**.

În figura 9.4 redăm istoria unui ciclu **IN** cu o singură stare **WAIT** impusă din exterior.

Ciclul OUT cu stări WAIT.

Din punctul de vedere al cererilor de așteptare, ciclul **OUT** nu diferă de ciclul **IN** (vezi figura 9.5)

Dacă semnalul $\overline{WAIT}$ permite sincronizarea procesorului cu evenimente externe prin mijloace hardware, instrucțiunea **HALT** oferă o posibilitate software cu scop asemănător.

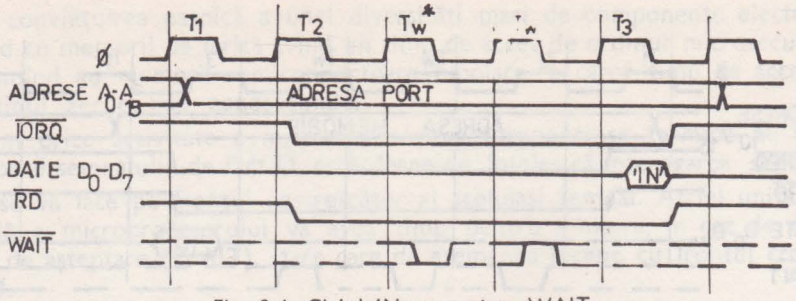


Fig. 9.4. Ciclul IN cu o stare WAIT

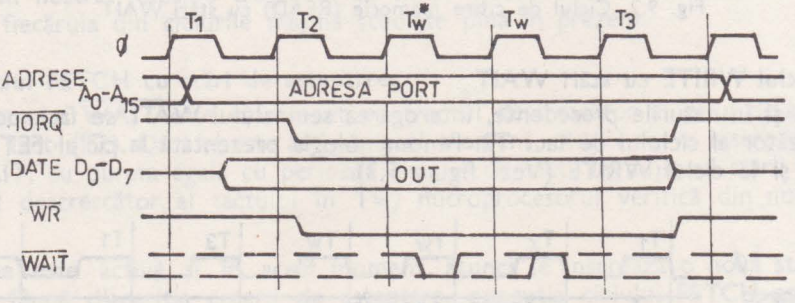


Fig. 9.5. Ciclul OUT cu o stare WAIT

9.2. Aventura a 8-a: oprirea (HALT)

Instrucțiunea **HALT** este destinată sincronizării execuției unui program cu evenimente externe, care prin natura lor sînt mai puțin legate de funcționarea microprocesorului, fiind din punctul de vedere al momentului de apariție, asincrone față de acesta. Ca și în cazul așteptărilor hardware (**WAIT**), s-a pornit de la considerentul că microprocesorul este cel mai rapid : execuția programului ajunge la prelucrarea instrucțiunii de oprire **HALT**, înainte ca evenimentul extern să fi avut loc. După execuția instrucțiunii **HALT** microprocesorul „se oprește", contorul program **PC** „înghețînd" la adresa instrucțiunii imediat următoare. La apariția evenimentului extern, sistemul generează o întrerupere (semnalul $\overline{\text{INT}}$), care va scoate microprocesorul din starea „amorțită" în care se afla după execuția instrucțiunii **HALT**.

După executarea rutinei de tratare a întreruperii, care într-un caz clasic de așteptare poate fi un simplu **RET**, microprocesorul continuă execuția programului din care fusese oprit, începînd cu instrucțiunea imediat următoare celei **HALT**.

Știind că întreruperile pot fi inhibate pe cale software, pentru ca ieșirea din starea **HALT** să se poată face la apariția cererii de întrerupere ($\overline{\text{INT}}$) este necesar ca înainte de a intra în **HALT** microprocesorul să execute o instrucțiune **EI** (**E**nable **I**nterrupt), de validare a întreruperilor. (Cererea de întrerupere nemascabilă — $\overline{\text{NMI}}$ — va fi oricum acceptată, indiferent de starea validată sau inhibată a sistemului de întreruperi).

Pentru cazul în care mult așteptatul eveniment nu mai are loc, dintr-un motiv sau altul, a fost prevăzută posibilitatea ca microprocesorul să poată fi scos

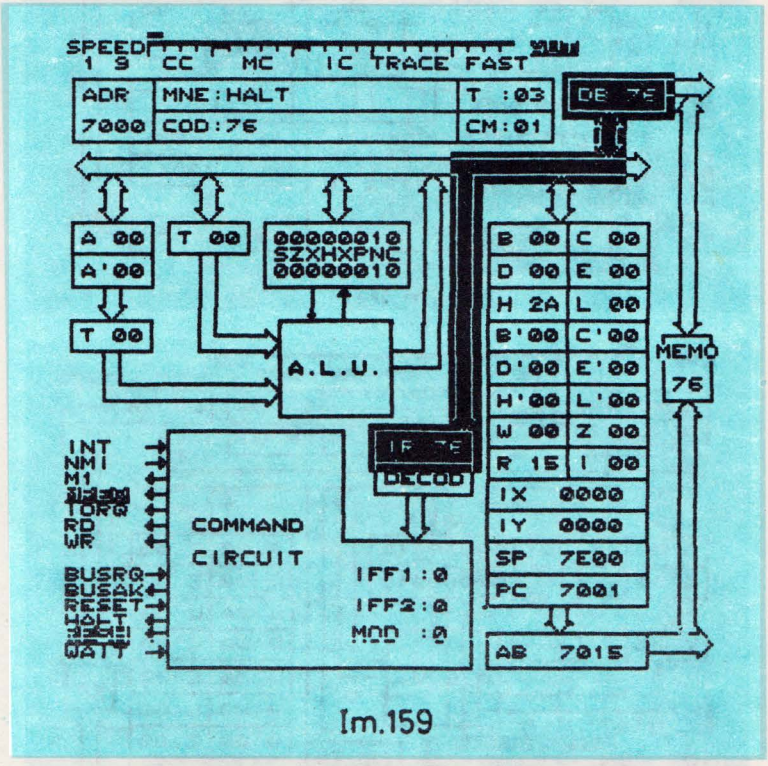
din starea oprită (HALT) și prin intermediul semnalului de ultimă instanță : $\overline{\text{RESET}}$.

Să urmărim cu ajutorul lui VISIBLE–Z80, într-o aventură de 12 imagini comportamentul microprocesorului în starea oprită (HALT) și abandonarea acestuia prin recepționarea semnalului $\overline{\text{RESET}}$.

```
7000            HALT            ;Im.159–Im.169  (CC)
0000      LD    SP, 4400        ;Im.170         (CC)
```

Instrucțiunea HALT se codifică pe 1 octet (76_H), și execuția ei durează 4 tacți procesor.

Începem cu CM=1 ;T=3. Codul (76_H) este depus în IR (Im.159) :

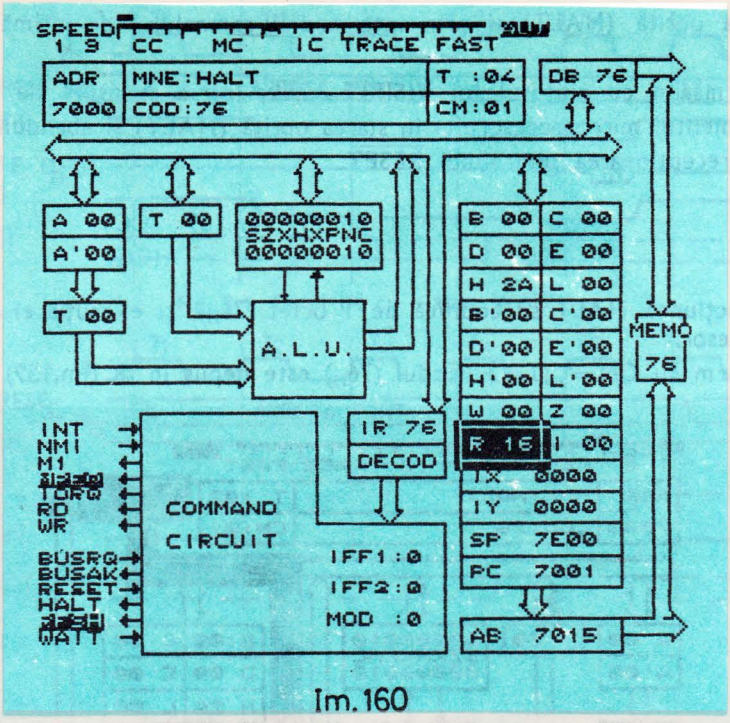


Im.159

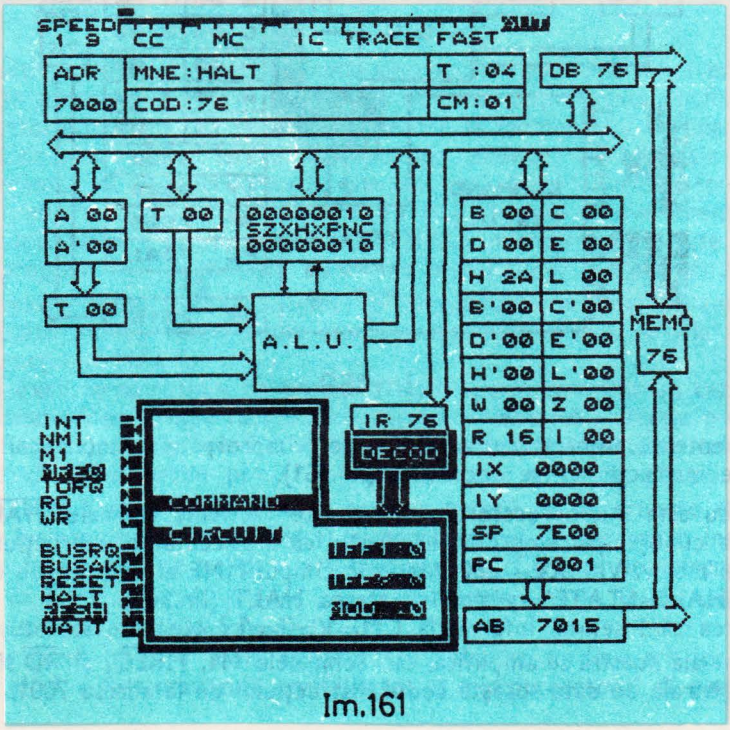
Se incrementează numărătorul adreselor de reîmprospătare R. (vă mai amintiți ?) După care instrucțiunea se decodifică (Im.161).

Și odată cu asta microprocesorul „se oprește" activînd semnalul $\overline{\text{HALT}}$ pentru a aduce la cunoștință celor din jur, faptul că a executat o instrucțiune HALT. Din același motiv, VISIBLE–Z80 înscrie în cîmpul MNE al colțului de comentarii mesajul „HALT STATE", semnalînd starea HALT (Im.162).

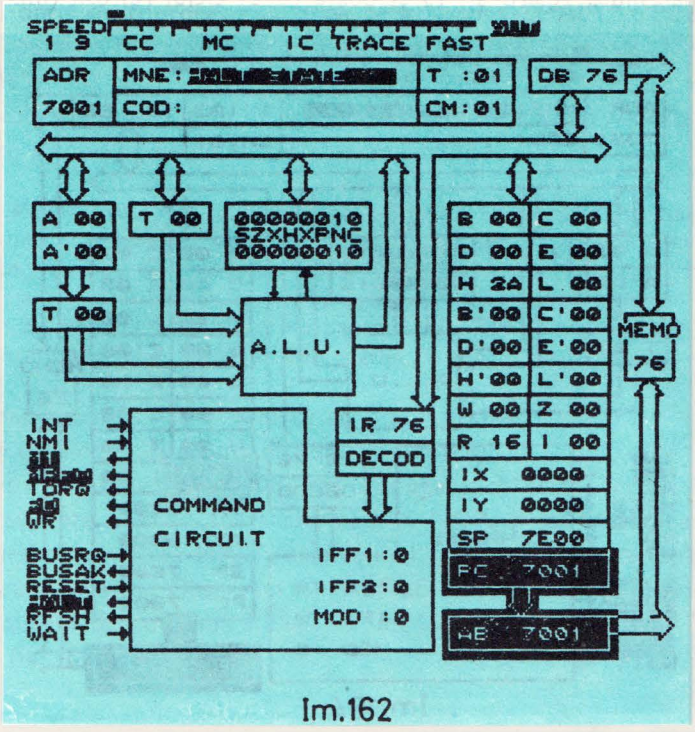
Dar ce vrea să însemne informația CM=1 ; T=1 ? Nu cumva, totuși ? Privind m.163 bănuiala noastră se amplifică, căci semnalele $\overline{\text{M1}}$, $\overline{\text{MREQ}}$, și $\overline{\text{RD}}$ sînt active, iar pe magistrala de date sosește codul instrucțiunii de la adresa 7001_H ($E5_H$).


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:HALT
COD:76
T :04
CM:01
DB 76
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 00
E 00
H 2A
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 16
I 00
IX 0000
IY 0000
SP 7E00
PC 7001
AB 7015
MEMO
76
IR 76
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

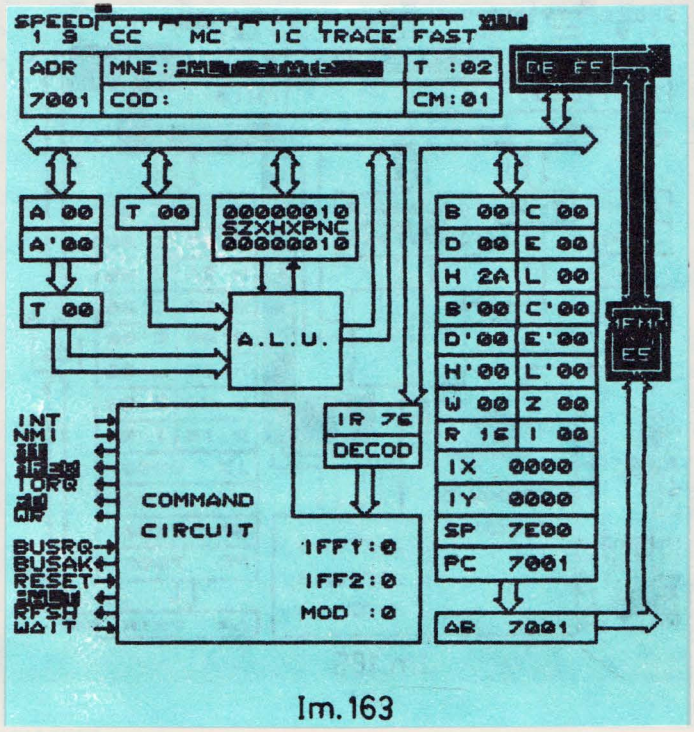
Im.160


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:HALT
COD:76
T :04
CM:01
DB 76
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 00
E 00
H 2A
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 16
I 00
IX 0000
IY 0000
SP 7E00
PC 7001
AB 7015
MEMO
76
IR 76
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
WAIT

Im.161

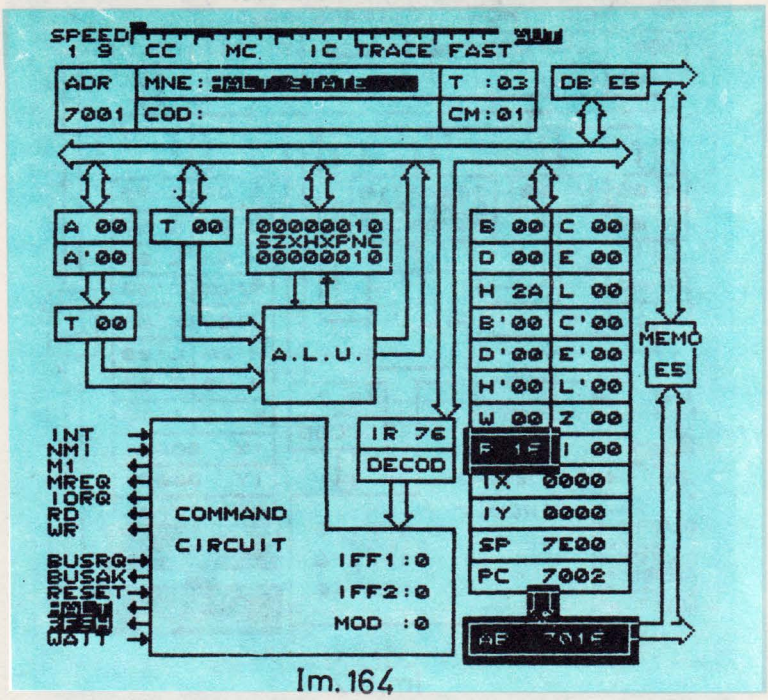

SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:
COD:
T :01
CM:01
DB 76
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00 C 00
D 00 E 00
H 2A L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 16 I 00
IX 0000
IY 0000
SP 7E00
PC 7001
AB 7001
MEMO
76
IR 76
DECOD
INT
NMI
TORQ
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
RFSH
WAIT

Im.162

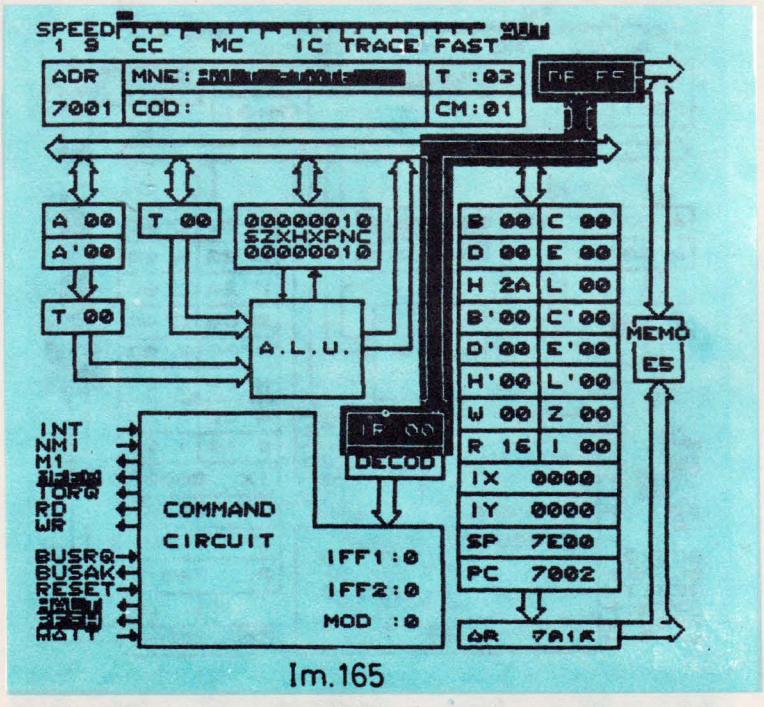

SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:
COD:
T :02
CM:01
DB FF
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00 C 00
D 00 E 00
H 2A L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 16 I 00
IX 0000
IY 0000
SP 7E00
PC 7001
AB 7001
IR 76
DECOD
INT
NMI
TORQ
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
BUSRQ
BUSAK
RESET
RFSH
WAIT

Im.163

Ba chiar mai mult. Apare și **T3** și odată cu el ciclul refresh (Im. 164).



Im. 164

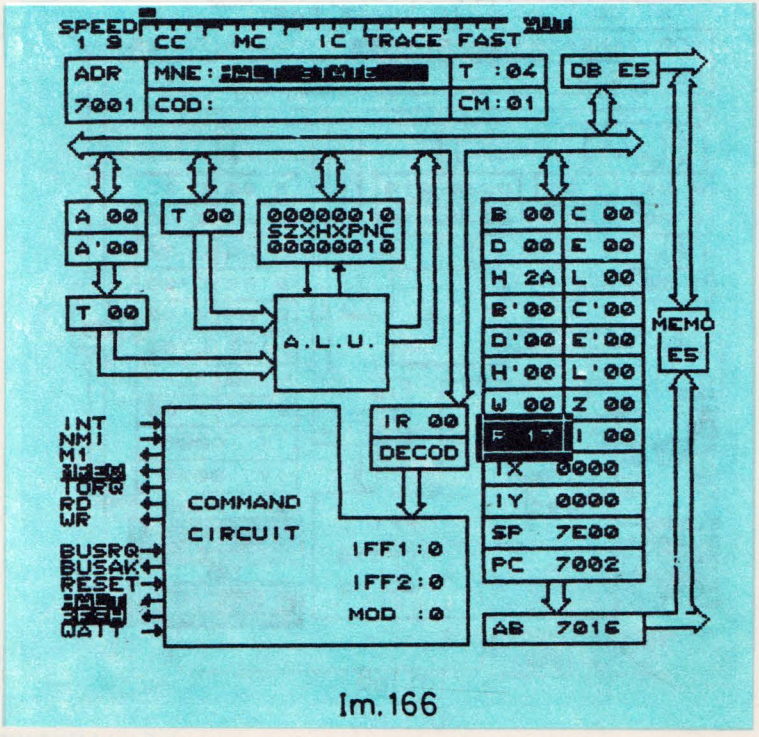


Im. 165

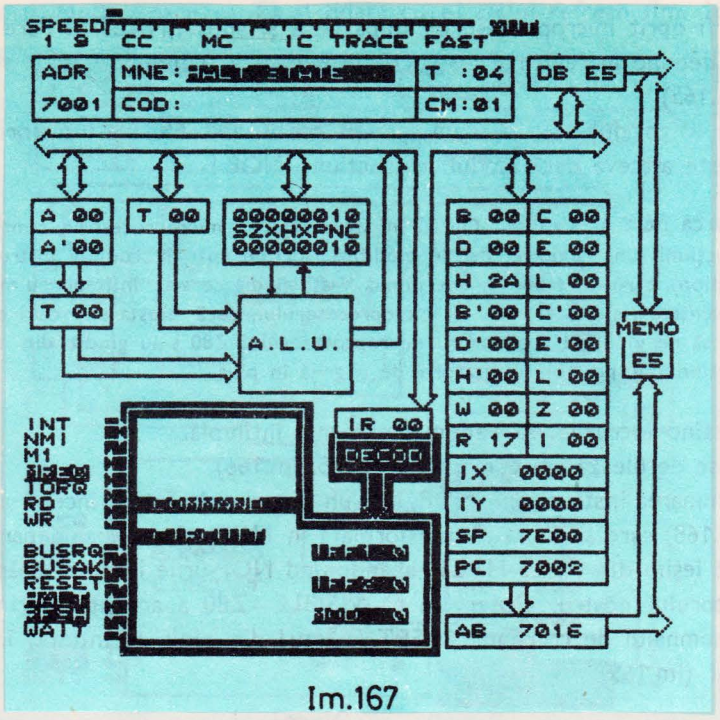
Să nu se fi oprit microprocesorul, după ce a și spus **„HALT" ?** Pare se că nu ! Codul se depune în registrul instrucțiune pentru a fi interpretat, și se activează **RFSH** (Im.165).
Ceva pare totuși dubios : în drum spre **IR** codul citit $E5_H$ s-a transformat în 00_H, care nu este altceva decît codul instrucțiunii **NOP** !

Remarca 56 : Asta a fost. „Oprit" în starea **HALT** microprocesorul continuă să execute instrucțiuni din memorie, dar le modifică, forțînd apariția codului instrucțiunii **NOP** (**NO** oPeration), care de fapt nu face nimic. V-ați gîndit ce s-ar întîmpla cu memoria dinamică încredințată ciclului refresh al microprocesorului, dacă acesta s-ar opri efectiv ? Nici nu e bine să ne gîndim ! Proiectanții microprocesorului **Z80** s-au gîndit din timp la acest aspect, scutindu-ne pe noi, utilizatorii de o grijă în plus.

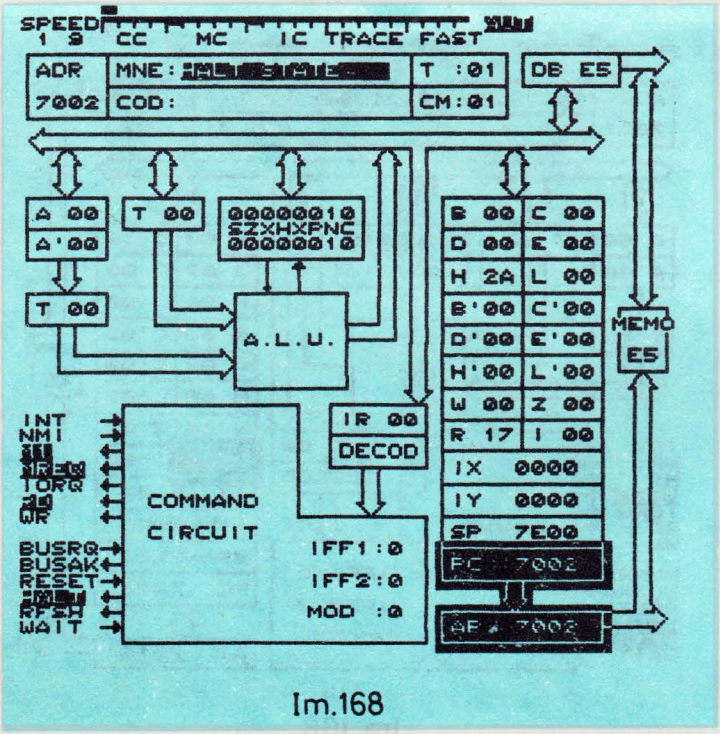
Așa stînd lucrurile, să vedem ce se mai întîmplă.
Execuția se derulează așa cum știm (Im.165, Im.166).
După terminarea instrucțiunii **NOP**, într-un singur ciclu fetch începe altă instrucțiune (Im.168) care și ea va fi transformată în **NOP**, și așa mai departe.
Dorind să ieșim din starea **HALT**, abandonînd NOP-urile infinite, apăsăm tasta **R** a calculatorului nostru. Observăm că **VISIBLE—Z80** a acceptat dorința noastră, activînd semnalul de comandă **RESET** la unul din pinii de intrare, în stînga jos a imaginii (Im.169).



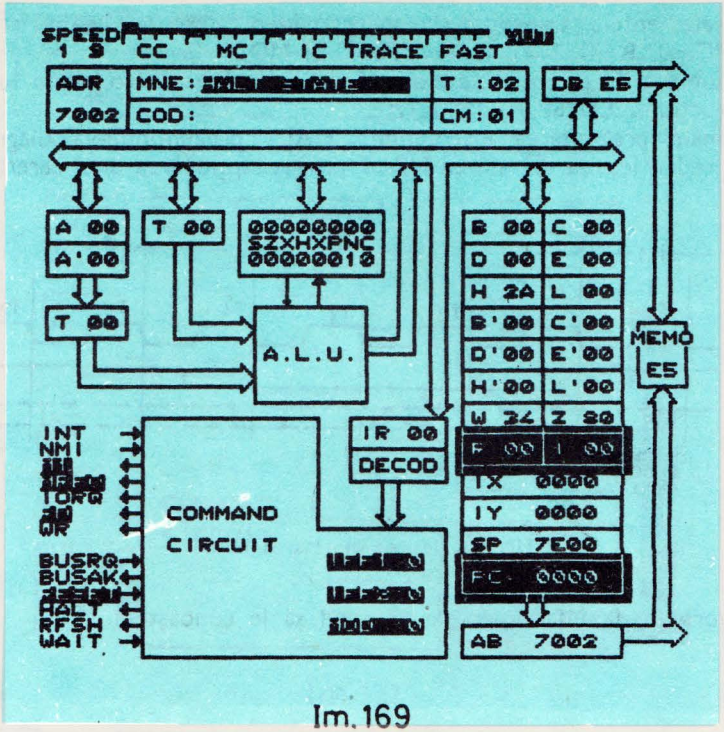
Im.166


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:
COD:
T :04
CM:01
DB E5
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 2A L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 17 I 00
IX 0000
IY 0000
SP 7E00
PC 7002
AB 7016
MEMO
E5
IR 00
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
WAIT

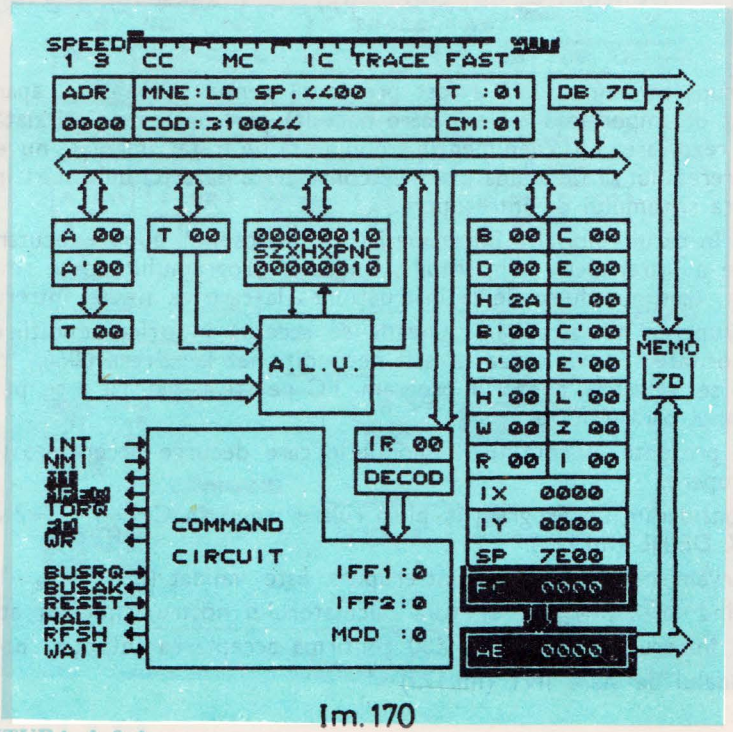
Im.167


SPEED
1 S CC MC IC TRACE FAST
ADR
7002
MNE:
COD:
T :01
CM:01
DB E5
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 00 E 00
H 2A L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 17 I 00
IX 0000
IY 0000
SP 7E00
PC 7002
AB 7002
MEMO
E5
IR 00
DECOD
INT
NMI
IORQ
WR
BUSRQ
BUSAK
RESET
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.168


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7002
MNE:
COD:
T :02
CM:01
DB EE
A 00
A'00
T 00
00000000
SZXHXPNC
00000010
B 00
C 00
D 00
E 00
H 2A
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 2A
Z 80
MEMO
E5
T 00
A.L.U.
IR 00
DECOD
R 00
I 00
IX 0000
IY 0000
SP 7E00
PC 0000
AB 7002
INT
NMI
IORQ
WR
BUSRQ
BUSAK
HALT
RFSH
WAIT
COMMAND
CIRCUIT

Im.169


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
0000
MNE:LD SP,4400
COD:310044
T :01
CM:01
DB 7D
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
B 00
C 00
D 00
E 00
H 2A
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 00
I 00
MEMO
7D
T 00
A.L.U.
IR 00
DECOD
IX 0000
IY 0000
SP 7E00
PC 0000
AE 0000
INT
NMI
IORQ
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0

Im.170

Urmărind elementele transpuse în invers video, observăm că el şi-a şi făcut efectul : **PC=0**, **R=0**, **I=0**, **IFF1=IFF2=0** şi **MOD=0**.

Aşa cum era de aşteptat, se execută saltul la adresa zero, unde începe execuţia instrucţiunii **LD SP,4400** (Im.170).

Terminăm prezentarea instrucţiunii **HALT** prezentîndu-vă diagramele de timp care redau ieşirea din starea **HALT** prin recepţionarea unei cereri de întrerupere.

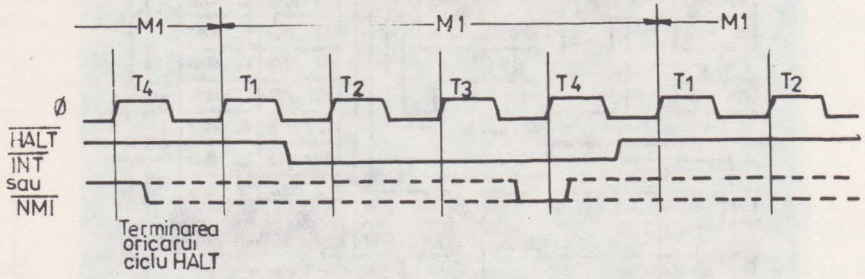


Fig. 9.6. Ieşirea din starea HALT

Tot vorbind de întreruperi, ar fi cazul să le cunoaştem.

9.3. Aventura a 9-a : întreruperea nemascabilă ($\overline{NMI}$)

Intreruperea nemascabilă a fost prevăzută pentru semnalarea apariţiei unui eveniment de importanţă majoră, care necesită participarea neîntîrziată a procesorului ca rezolvarea lui. Evenimentul trebuind să fie tratat imediat, nu este admisibil la cererea lui să fie amînată sau desconsiderată datorită unei stări întîmplător inhibate ca sistemului de întrerupere.

Ca şi în cazul celorlalte întreruperi (cele mascabile) după executarea rutinei de tratare a întreruperii nemascabile, execuţia programului poate reveni în cel abandonat, continuîndu-se de la instrucţiunea la care ea fusese întreruptă.

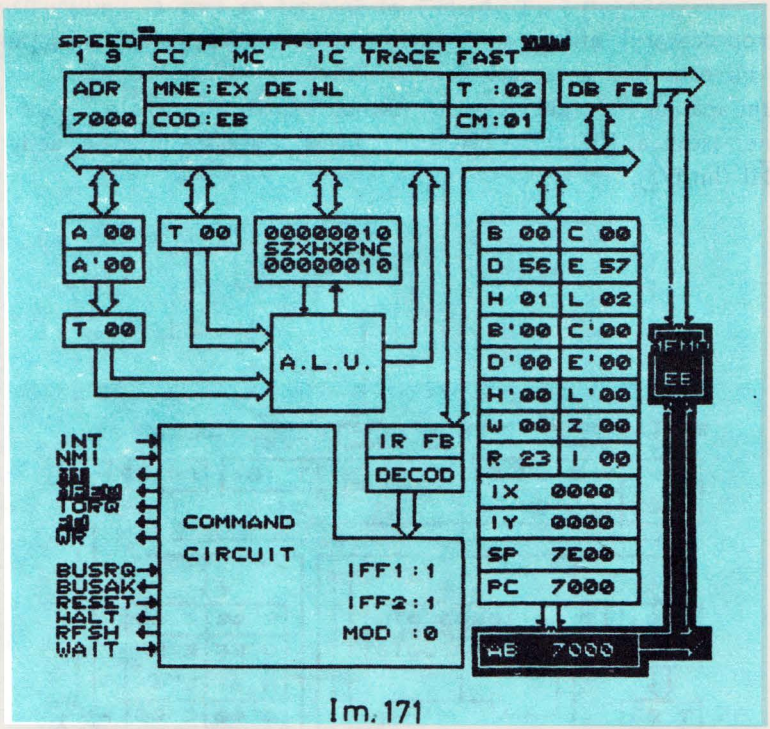
Intreruperea nemascabilă, spuneam, se acceptă în orice condiţii, cu cea mai mare prioritate. Ea va cauza un salt necondiţionat la adresa 0066_H, după ce în prealabil se salvează contorul program **PC** pe stivă, pentru a se putea reveni la programul părăsit forţat.

Vom prezenta în continuare modul în care decurge acceptarea unei cereri de întrerupere.

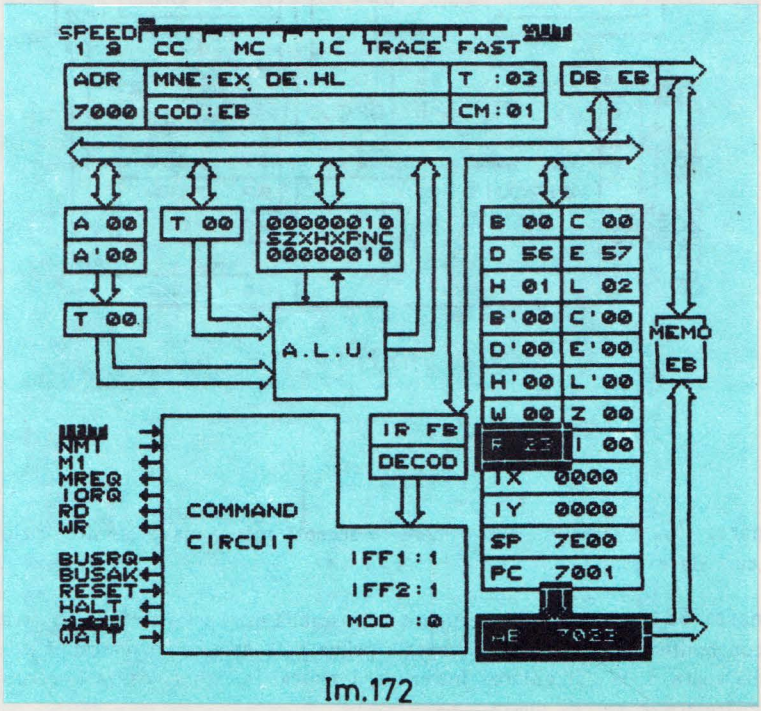
Să considerăm un program în plină rulare, ajuns în CM=1, T=2 al instrucţiunii **EX DE,HL** (Im.171).

Observăm că sistemul de întrerupere este validat ($IFF_1=IFF_2=1$)

Apăsînd tasta „I" pe claviatura calculatorului nostru, simulăm apariţia unei cereri de întreruperi. **VISIBLE – Z80** confirmă acceptarea intenţiei noastre, activînd semnalul de stare $\overline{INT}$ (Im.172)


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:EX DE,HL
COD:EB
T :02
CM:01
DB FB
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 56
E 57
H 01
L 02
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 23
I 00
IX 0000
IY 0000
SP 7E00
PC 7000
MEMO
EB
AB 7000
IR FB
DECOD
INT
NMI
IORQ
WR
COMMAND
CIRCUIT
IFF1:1
IFF2:1
MOD :0
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT

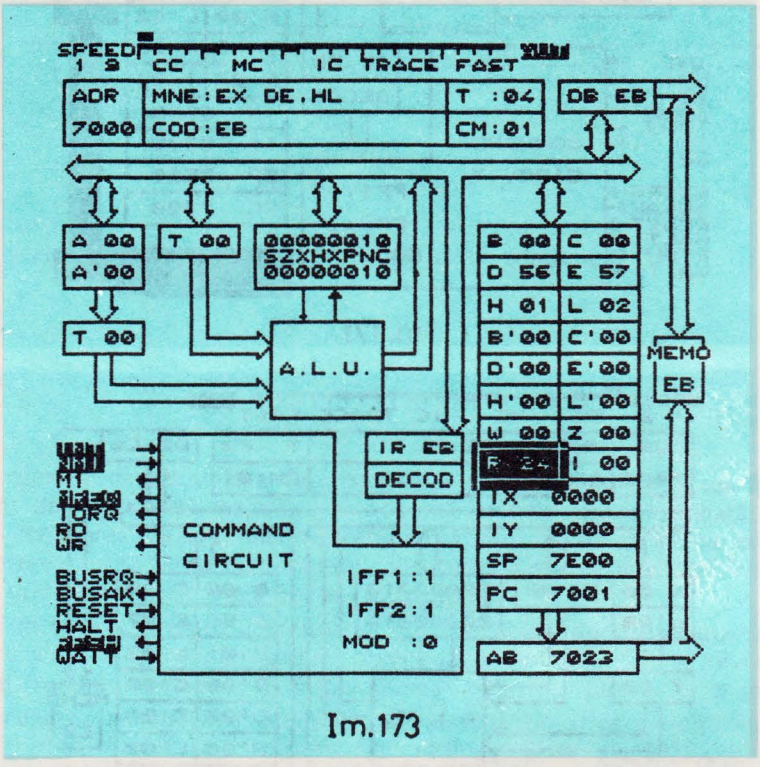
Im.171


SPEED
1 9
CC
MC
IC
TRACE
FAST
ADR
7000
MNE:EX DE,HL
COD:EB
T :03
CM:01
DB EB
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 00
C 00
D 56
E 57
H 01
L 02
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 23
I 00
IX 0000
IY 0000
SP 7E00
PC 7001
MEMO
EB
IR FB
DECOD
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
IFF1:1
IFF2:1
MOD :0
BUSRQ
BUSAK
RESET
HALT
WAIT

Im.172

Microprocesorul aflîndu-se în mijlocul executării unei instrucțiuni, întreruperea nu este încă acceptată.

Acum apăsăm tasta „N" a calculatorului, pentru a genera o cerere de întrerupere nemascabilă. VISIBLE—Z80 ne confirmă și această intenție, activînd semnalul $\overline{NMI}$ (Im.173).

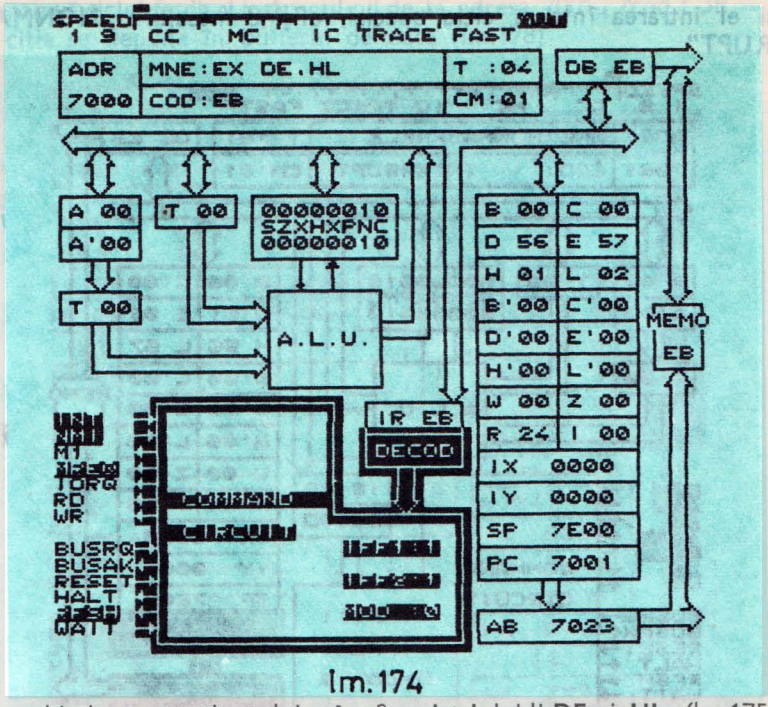


Im.173

Remarca 57. Cererea de întrerupere se acceptă totdeauna la sfîrșitul ciclului *instrucțiune*, în curs de execuție.

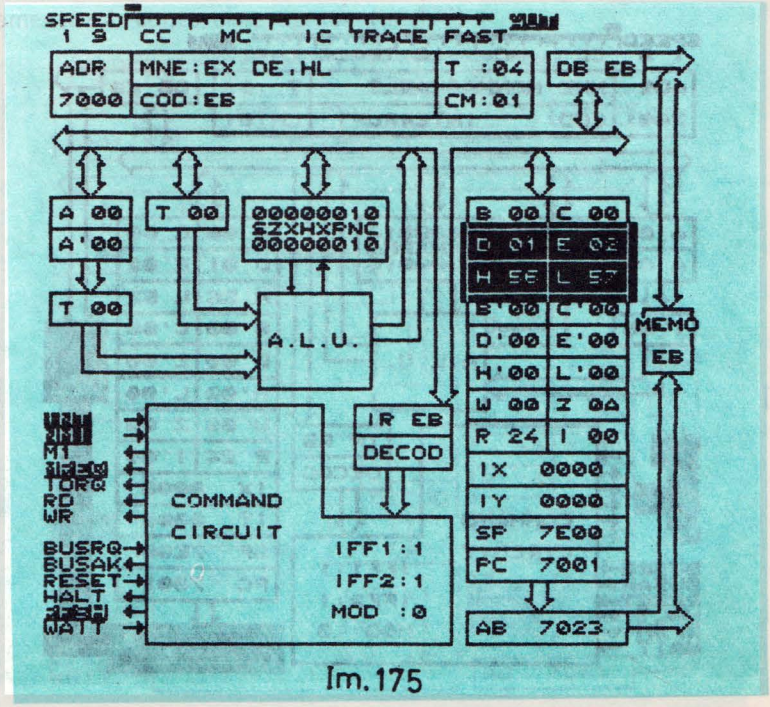
Remarca 58 : Cererea de întrerupere nemascabilă este prioritară celei mascabile : dacă ele apar concomitent, cea care este acceptată prima dată este cea nemascabilă ; dacă cererea nemascabilă apare după acceptarea întreruperii mascabile, atunci ea va întrerupe execuția rutinei de tratare a cererii mascabile.

Execuția instrucțiunii în curs se termină în CM=1, T=4 (Im.174),



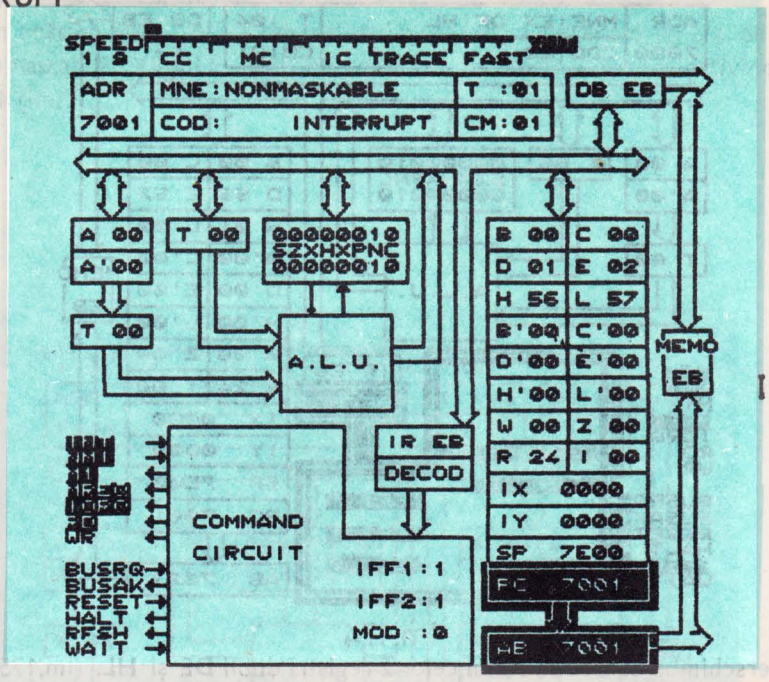
Im.174

prin interschimbarea conținutului celor 2 regiștri dubli DE și HL. (Im.175).

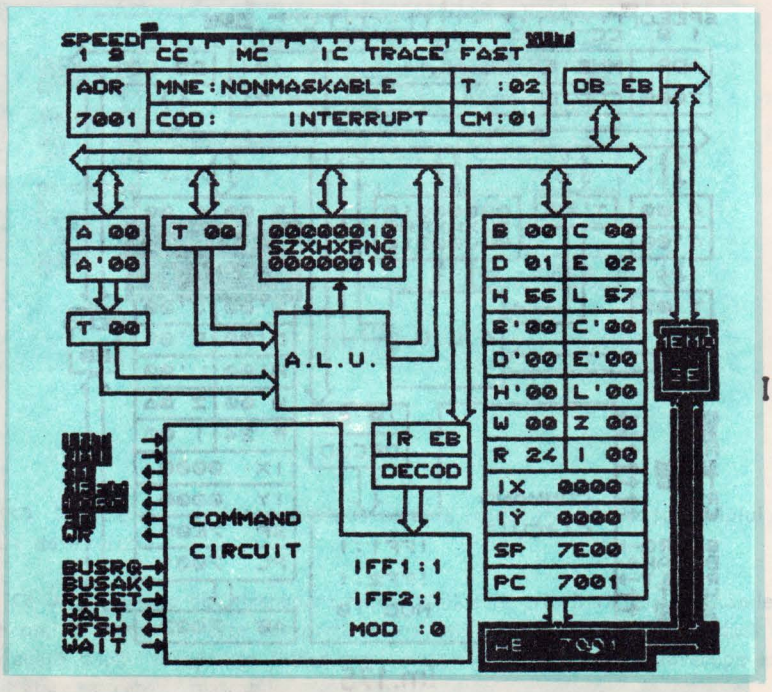


Im.175

Iar acum este acceptată cererea de întrerupere (Im.176). **VISIBLE—Z80** ne confirmă și el intrarea într-un ciclu special, afișînd mesajul : **„NONMASKABLE INTERRUPT"**

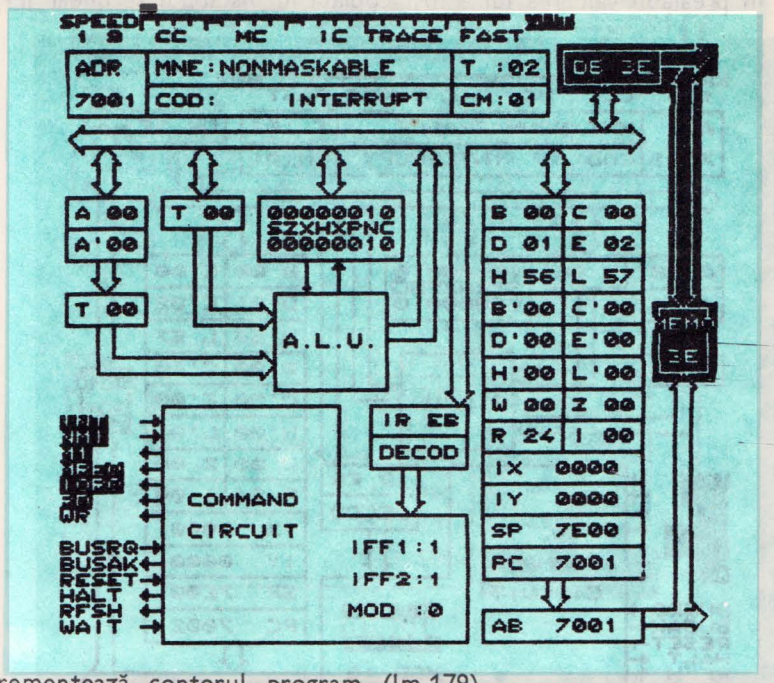


Im. 176



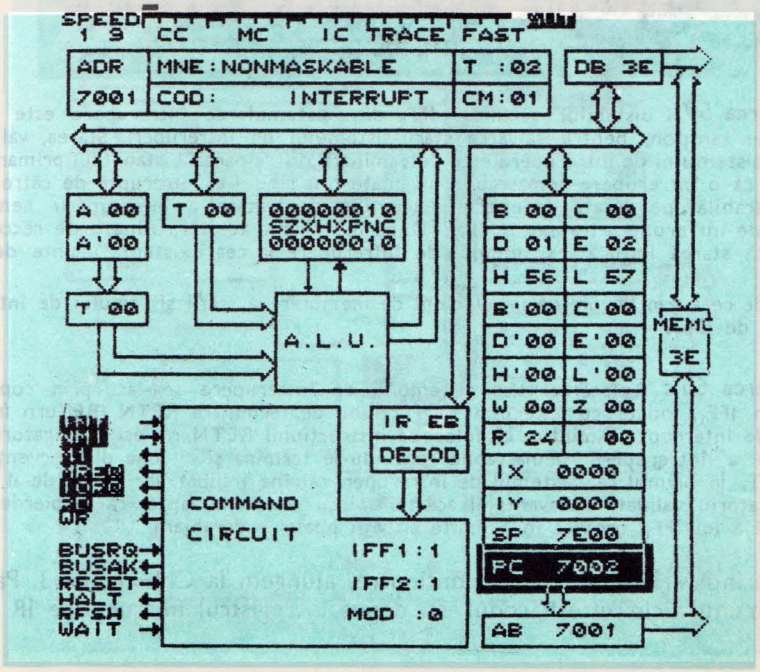
Im. 177

Urmărind în continuare evoluția microprocesorului, constatăm că deocamdată se continuă ciclul fetch al instrucțiunii de la adresa 7001_H (cod : 3E = LD A,n) Octetul citit se depune în bufferul de date (Im.178)



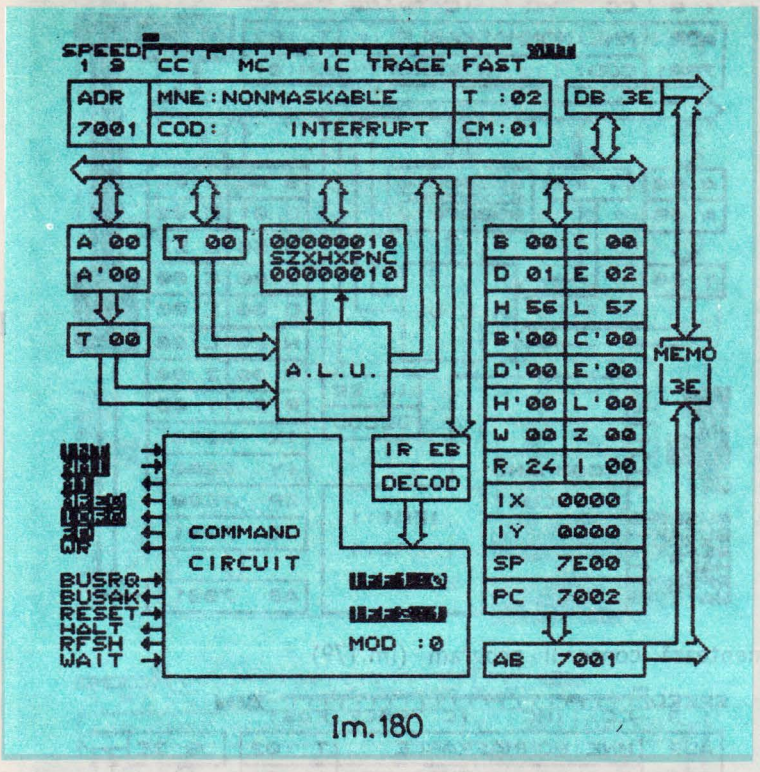
Im.178

Se incrementează contorul program (Im.179)



Im. 179

De abia acum (CM=1, T=2) apare un element nou, distinct față de un fetch obișnuit : bistabilul primar de stare IFF_1 al sistemului de întrerupere se șterge, după ce în prealabil valoarea lui a fost copiată în bistabilul secundar IFF_2 :



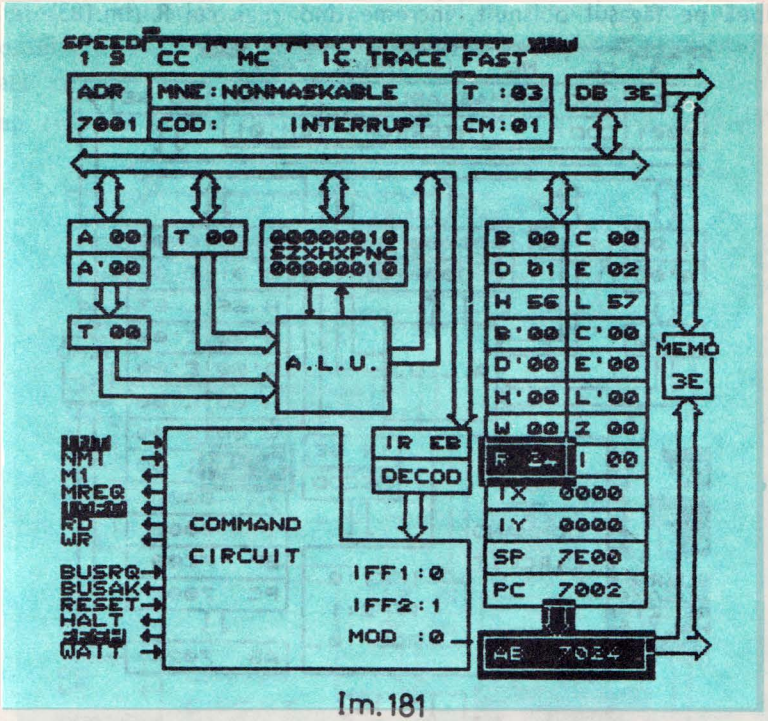
Im.180

Remarca 59 : Bistabilul secundar IFF_2 din sistemul de întrerupere este o celulă de memorie tampon, pentru salvarea stării sistemului de întreruperi. Starea, validată sau inhibată, a sistemului de întrerupere este determinată de valoarea bistabilului primar de stare IFF_1. Știind că o întrerupere nemascabilă nu poate fi la rîndul ei întreruptă de către o întrerupere mascabilă, pe durata de execuție a rutinei de tratare a întreruperii nemascabile, sistemului de întrerupere trebuie inhibat. După terminarea acestei rutine, este recomandabil să se refacă starea inițială a sistemului de întrerupere la cea existentă înainte de apariția $\overline{NMI}$.

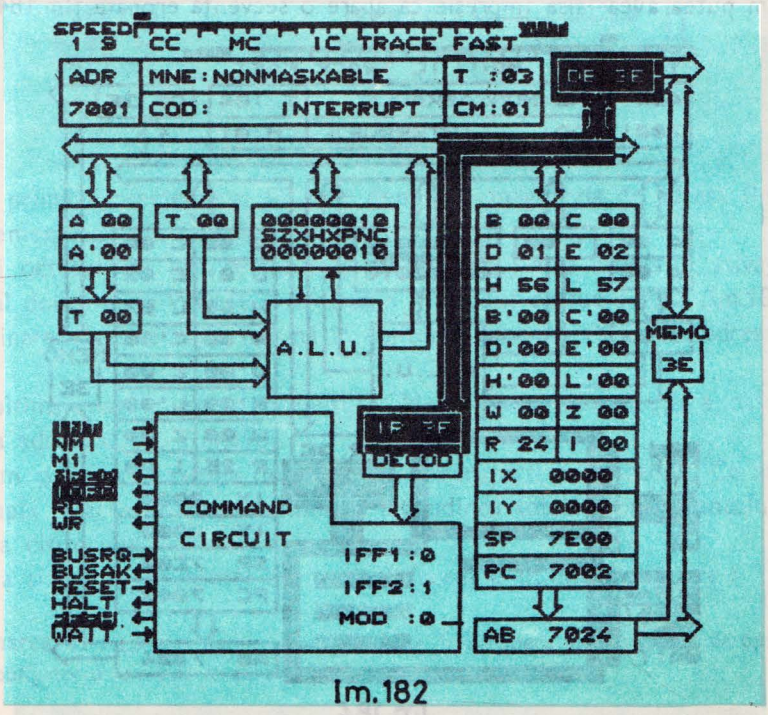
Iată de ce avem nevoie de un element de memorare a stării sistemului de întrerupere, rol preluat de IFF_2.

Remarca 60 : Refacerea stării sistemului de întrerupere se face prin copierea lui lui IFF_2 în IFF_1, odată cu execuția instrucțiunii de terminare **RETN** (**RET**urn from **N**on — maskable interrupt). Amintim că folosirea instrucțiunii **RETN** nu este obligatorie, rutina de tratare a întreruperii nemascabile putîndu-se termina și cu una din secvențele **RET** sau **EI, RET.** În primul caz sistemul de întreruperi rămîne inhibat, iar în cel de-al doilea el va fi obligatoriu validat. Dezavantajul acestei soluții constă în faptul că se pierde valoarea programată a lui IFF_1, ceea ce în anumite situații poate fi deranjant.

Urmărind **VISIBLE—Z80** în continuare, ajungem la CM=1, T=3. Paralel cu demararea unui ciclu refresh, codul este depus în registrul instrucțiune **IR** (Im.181, Im.182)

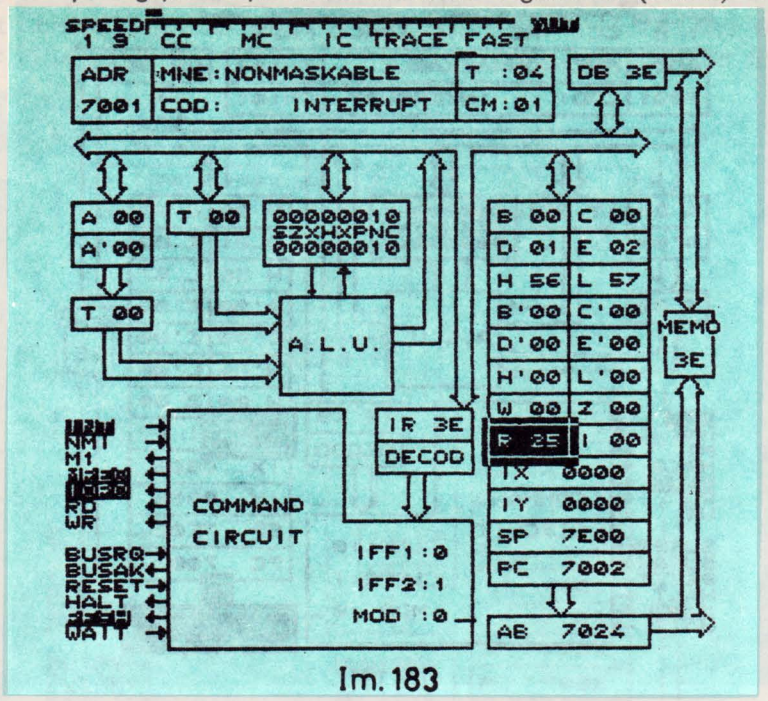

SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:NONMASKABLE
COD: INTERRUPT
T :03
CM:01
DB 3E
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 24 I 00
IX 0000
IY 0000
SP 7E00
PC 7002
MEMO
3E
IR EB
DECOD
NMI
M1
MREQ
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:1
MOD :0
BUSRQ
BUSAK
RESET
HALT
WAIT
AB 7024

Im.181


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:NONMASKABLE
COD: INTERRUPT
T :03
CM:01
DB 3E
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 24 I 00
IX 0000
IY 0000
SP 7E00
PC 7002
MEMO
3E
IR 3E
DECOD
NMI
M1
RD
WR
COMMAND
CIRCUIT
IFF1:0
IFF2:1
MOD :0
BUSRQ
BUSAK
RESET
HALT
WAIT
AB 7024

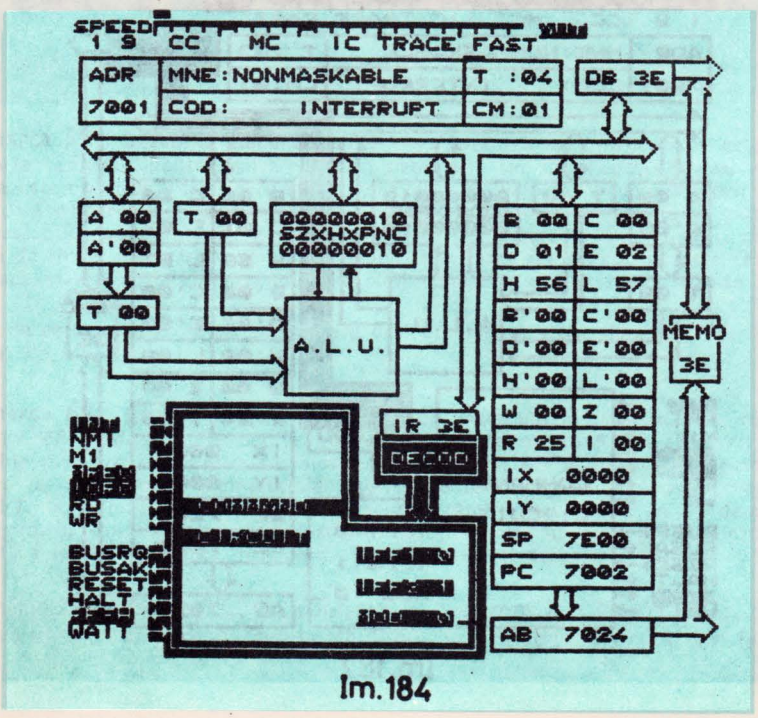
Im.182

Se continuă pe făgaşul obişnuit, incrementînd registrul R (Im.183)



Im. 183

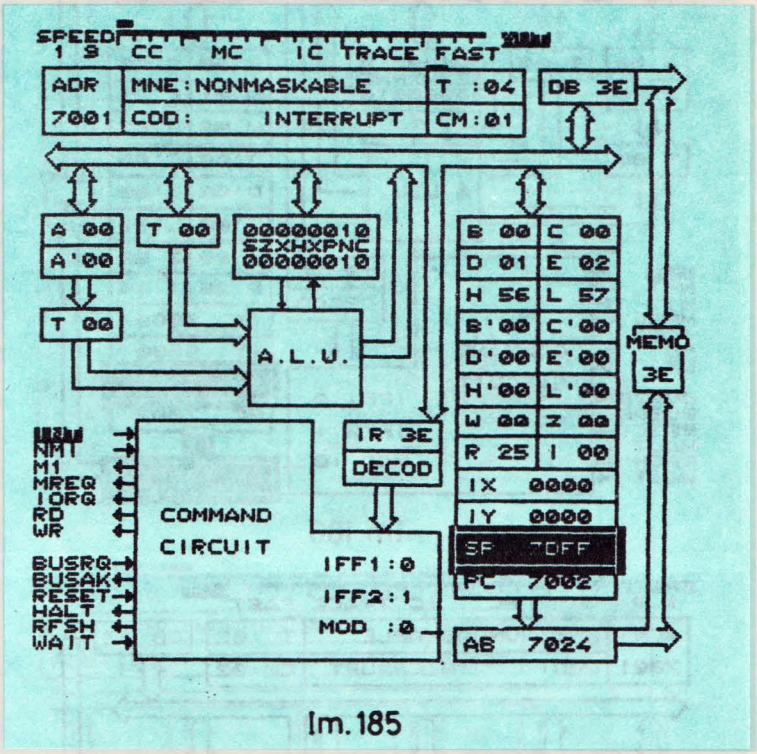
Acum am putea avea falsa impresie că apare o secvenţă eronată (Im.184)



Im. 184

Dar unitatea de comandă a microprocesorului nu greșește : rezultatul decodificării codului citit — $3E_H$ în cazul de față — este ignorat, elementul apelat în continuare nefiind **PC**, ci **SP** !

Încep ciclurile mașină destinate salvării adresei de întoarcere : (Im.185)

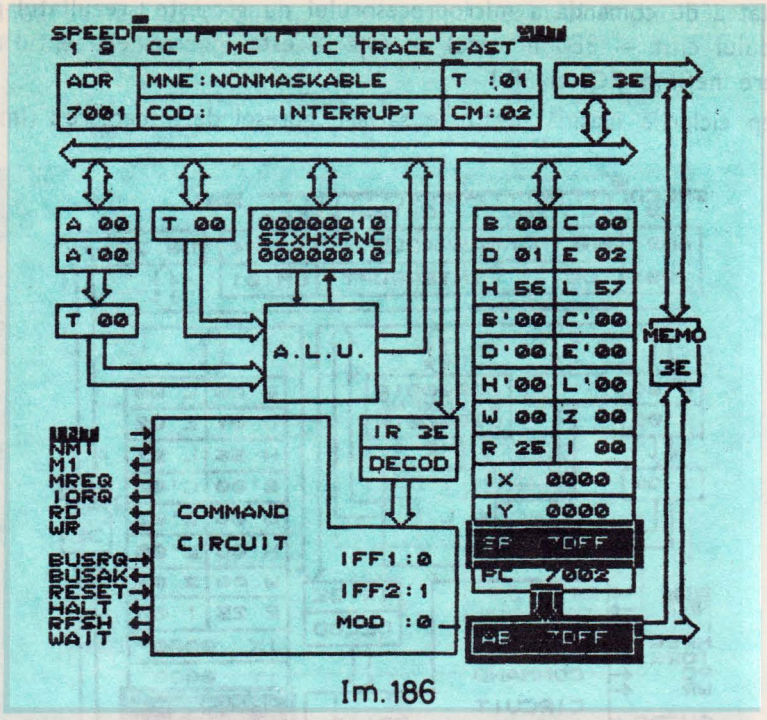


Im.185

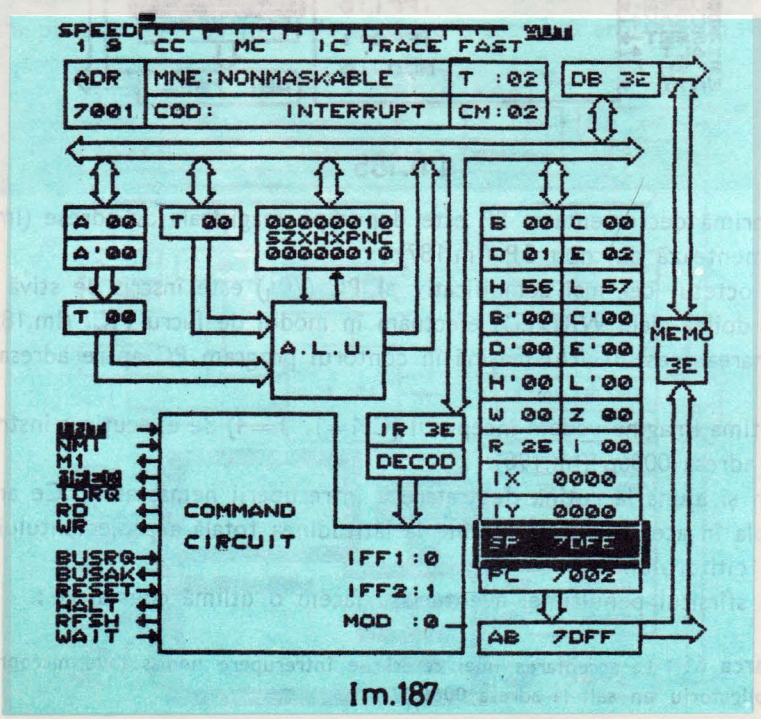
După o primă decrementare, **SP** este depus pe magistrala de adrese (Im.186)
Se decrementează din nou **SP** (Im.187)
Iar apoi octetul cel mai semnificativ al **PC** (70_H) este înscris pe stivă (Im.188)
Cel de-al doilea ciclu **WRITE** îl efectuăm în modul de lucru **MC.** (Im.189).
La terminarea acestui ciclu mașină în contorul program **PC** apare adresa de salt **(0066_H)**

În ultima imagine vedem începutul (CM=1, T=4) de execuție a instrucțiunii locate la adresa 0066_H (Im.190).
Astfel am și ajuns la rutina de tratare a întreruperii nemascabile. Ce anume se va întîmpla în această rutină rămîne la latitudinea totală a proiectantului de sistem și a cititorului.
Ajunși la sfîrșitul penultimei aventuri să facem o utlimă constatare :

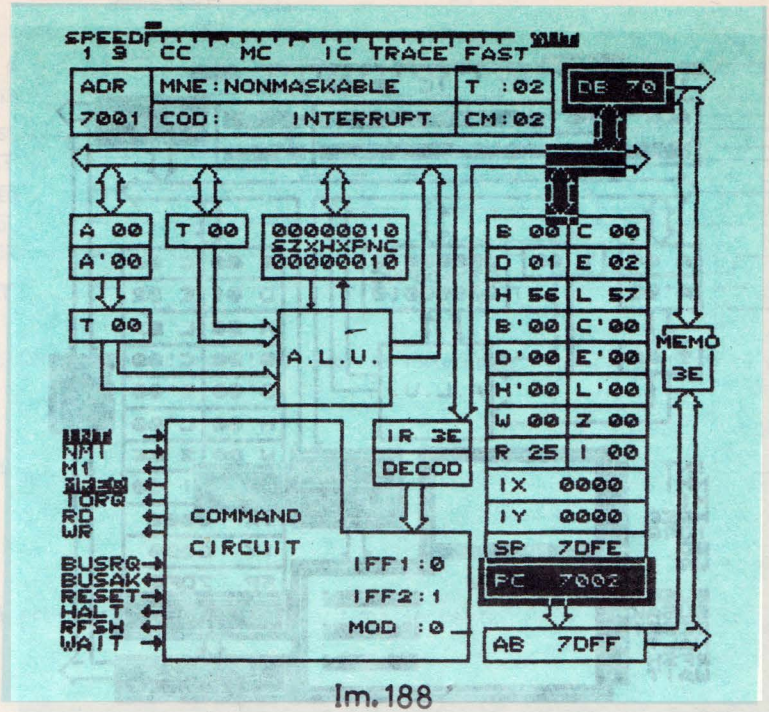
Remarca 61 : La acceptarea unei cereri de întrerupere nemascabile microprocesorul execută obligatoriu un salt la adresa **0066_H**.


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:NONMASKABLE
COD: INTERRUPT
T :01
CM:02
DB 3E
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 25 I 00
IX 0000
IY 0000
SP 7DFF
PC 7002
MEMO
3E
T 00
A.L.U.
IR 3E
DECOD
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:1
MOD :0
AB 7DFF

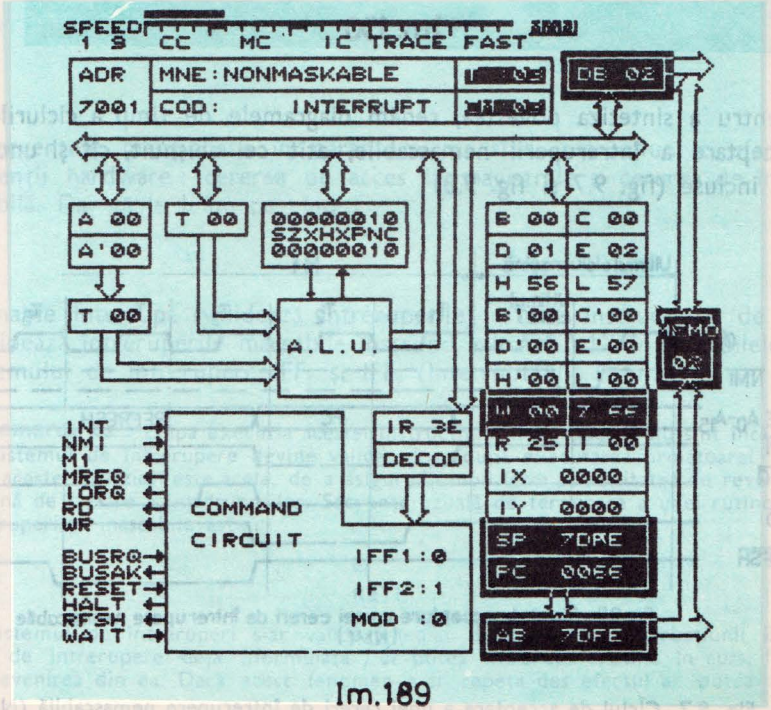
Im.186


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE:NONMASKABLE
COD: INTERRUPT
T :02
CM:02
DB 3E
A 00
A'00
T 00
00000010
SZXHXPNC
00000010
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 25 I 00
IX 0000
IY 0000
SP 7DFE
PC 7002
MEMO
3E
T 00
A.L.U.
IR 3E
DECOD
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:1
MOD :0
AB 7DFF

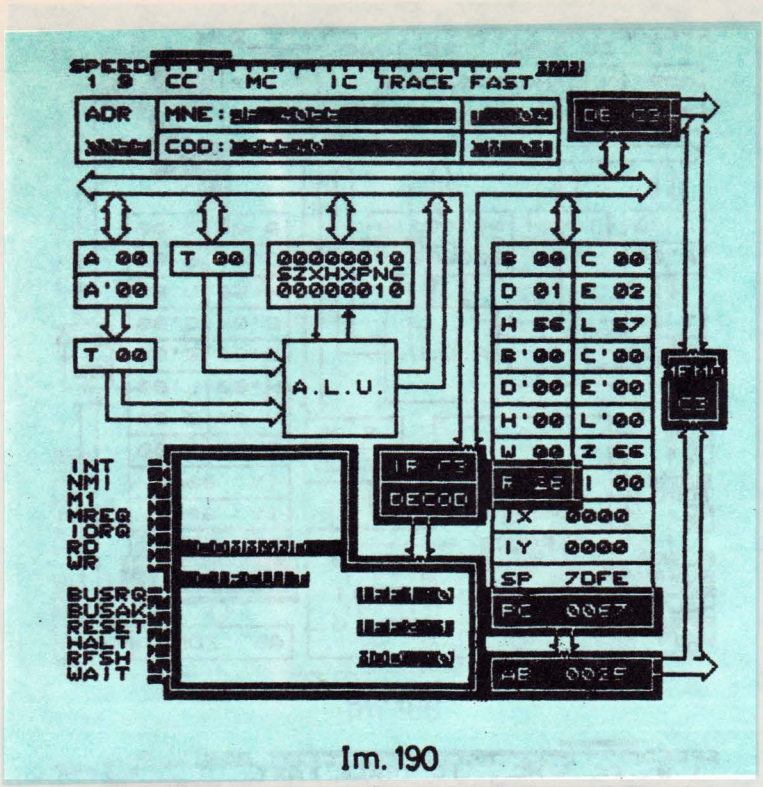
Im.187


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE: NONMASKABLE
COD: INTERRUPT
T :02
CM:02
DB 70
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 25 I 00
IX 0000
IY 0000
SP 7DFE
PC 7002
MEMO
3E
IR 3E
DECOD
NMI
M1
IORQ
RD
WR
COMMAND
CIRCUIT
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
IFF1:0
IFF2:1
MOD :0
AB 7DFF

Im.188


SPEED
1 S CC MC IC TRACE FAST
ADR
7001
MNE: NONMASKABLE
COD: INTERRUPT
DB 02
A 00
A'00
T 00
T 00
00000010
SZXHXPNC
00000010
A.L.U.
B 00 C 00
D 01 E 02
H 56 L 57
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z FF
R 25 I 00
IX 0000
IY 0000
SP 7DFE
PC 00EE
MEMO
02
IR 3E
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
COMMAND
CIRCUIT
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
IFF1:0
IFF2:1
MOD :0
AB 7DFE

Im.189



Im. 190

Pentru a sintetiza povestea, redăm diagramele de timp a ciclurilor mașină de acceptare a întreruperii nemascabile, atît cel obișnuit, cît și unul cu stări WAIT incluse (fig. 9.7 și fig. 9.8)

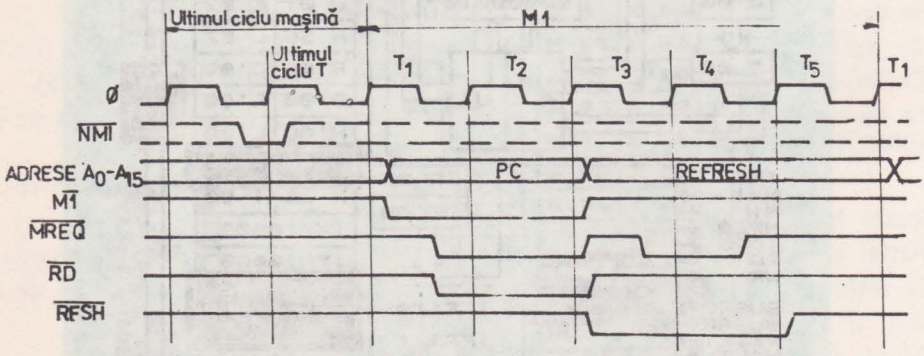


Fig.9.7. Ciclul de acceptare a unei cereri de întrerupere nemascabile (NMI)

Fig. 9.7. Ciclul de acceptare a unei cereri de întrerupere nemascabilă (NMI)

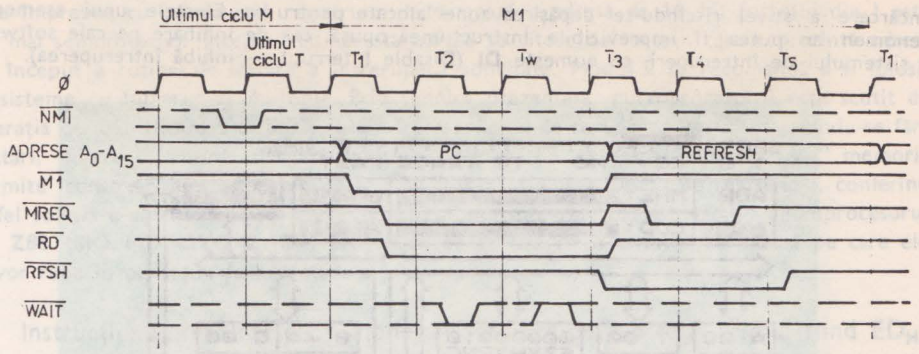


Fig. 9.8. Ciclul de acceptare a unei cereri de întrerupere nemascabilă (NMI) cu stare WAIT

9.4. Aventura a 10-a: întreruperile mascabile și accesul la magistrale ($\overline{INT}$, $\overline{BUSRQ}$, $\overline{BUSAK}$)

Vă propunem să urmăriți o ultimă secvență :

```
7000        EI                  ; Im.191                (MC)
7001        IM2                 ; Im.192 – Im.193       (MC)
7003        LD      A,B         ; Im.194 – Im.198       (MC)
(BUSRQ-BUSAK)                   ; Im.199
(INT)                           ; Im.200 – Im.216
00E5        LD      HL,0006     ; Im.217                (MC)
```

Secvențele menționate în paranteze nu sînt instrucțiuni ci efectele unor intervenții hardware : cererea de acces la magistrale și cererea de întrerupere mascabilă. Dar să le luăm pe rînd :

```
7000        EI                  ;Im.191                 (MC)
```

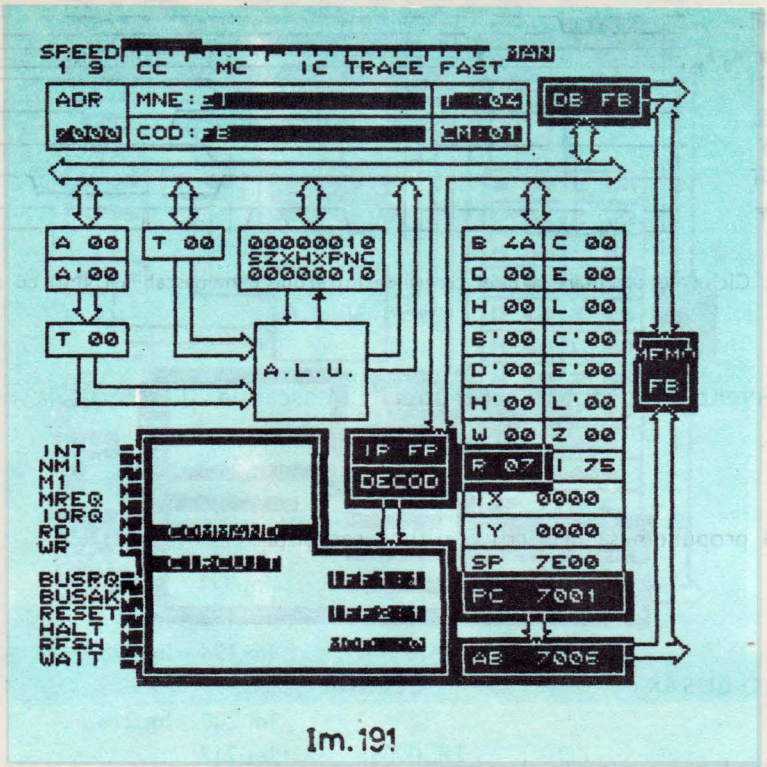
Enable Interrupt (validează întreruperile) este o instrucțiune de comandă. Ea validează întreruperile mascabile înscriind valoarea „1" în bistabilele de stare a sistemului de întrerupere IFF_1 și IFF_2 (Interrupt Flip-Flop).

Remarca 62 : După execuția acestei instrucțiuni, întreruperile nu sînt încă acceptate ! Fizic, sistemul de întrerupere devine validat doar după executarea următoarei instrucțiuni. Scopul acestei întîrzieri este acela, de a asigura în mod ferm posibilitatea de revenire dintr-o subrutină de tratare a întreruperilor. Secvența uzuală de terminare a unei rutine de tratare a întreruperilor mascabile este :

```
EI
RETI
```

Dacă sistemul de întreruperi s-ar valida imediat după execuția instrucțiunii **EI**, atunci o cerere de întrerupere deja „formulată", ar putea întrerupe rutina în curs, fără să se fi făcut revenirea din ea. Dacă acest fenomen s-ar repeta des efectul ar putea fi o supra-

încărcare a stivei riscîndu-se depășirea zonei alocate pentru ea. Efectele unui asemenea fenomen ar putea fi imprevizibile. Instrucțiunea opusă, cea de inhibare pe cale software a sistemului de întreruperi, se numește **DI** (Disable Interrupt — inhibă întreruperea).



Im.191

Astfel se programează cel mai complex (și performant) mod de întrerupere al microprocesorului **Z80**.

Remarca 63 : Cele 3 moduri de întrerupere **IM 0, IM 1** și **IM 2** diferă esențial în ceea ce privește tehnicile pe care le folosesc.

În modul **0**, dispozitivul de intrare/ieșire care a solicitat întreruperea, va trebui să furnizeze în ciclul de acceptare a întreruperii, codul unei instrucțiuni. Această instrucțiune va fi, așa cum probabil vă imaginați, o instrucțiune de apel a unei subrutine. (Nu va fi instrucțiunea de salt, căci trebuie salvat conținutul contorului program **PC**).

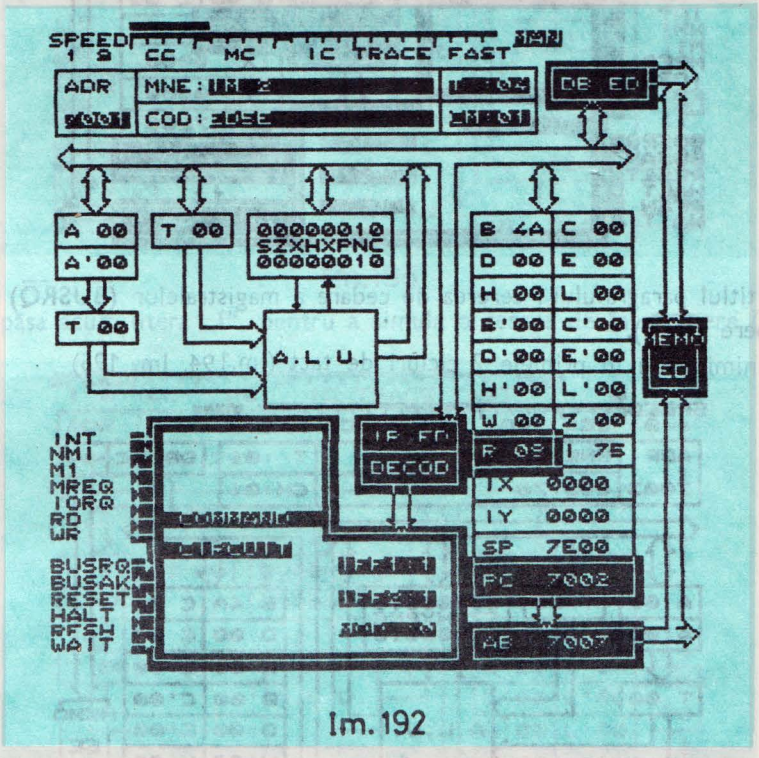
De cele mai multe ori proiectanții preferă instrucțiunile **RST** (**ReSTart**) fiindcă ele sînt codificate pe un octet. Dacă se folosește o instrucțiune de tip **CALL**, atunci dispozitivul I/E întrerupător va trebui să furnizeze 3 octeți, îngreunînd sarcina proiectantului de hardware. Folosind instrucțiunea **RST** saltul se va face la una din adresele **(0, 8, 10H,...,38H)**. Dezavantajul acestui tip de apel este că rutinele vor trebui să fie foarte scurte (8 byte), sau execuția lor se va lungi cu o instrucțiune de salt suplimentară. Dacă se generează codul instrucțiunii **CALL**, atunci devine accesibilă orice adresă de memorie, dar apare dezavantajul amintit. **Modul 0** este echivalat cu unicul mod de întrerupere a procesorului **I8080**.

În **modul 1, Z80** execută automat o instrucțiune **RST 38H**. Acest mod se recomandă a fi folosit în sistemele cu o singură sursă de întrerupere.

În **modul 2**, întreruperea este vectorizată. Folosind modul 2, în ciclul de acceptare a unei întreruperi, perifericul apelant va pune la dispoziția unității centrale un octet de adresă,

care împreună cu conținutul registrului **I** formează o adresă de 16 bit (octetul din **I** este cel mai semnificativ). Începînd cu această adresă se citesc doi octeți, care reprezintă adresa de început a rutinei de tratare a întreruperii solicitate. Modul 2 se recomandă a fi folosit în sisteme cu întreruperi multiple. Prin tehnica prezentată, microprocesorul este scutit de operația de identificare a dispozitivului care a solicitat întreruperea, saltul efectuîndu-se fără căutări, direct în rutina aferentă acestuia. Tehnica de adresare indirectă „via" memorie permite construirea unei tabele de salturi într-o pagină adresată de registrul **I**, conferind astfel software-ului eficiență și eleganță. Circuitelor integrate **LSI** din familia microprocesorului **Z80 (SIO, PIO, CTC, DART,** etc.) li se poate programa octetul de adresă pe care ele îl vor reda în procesul de acceptare a întreruperii.

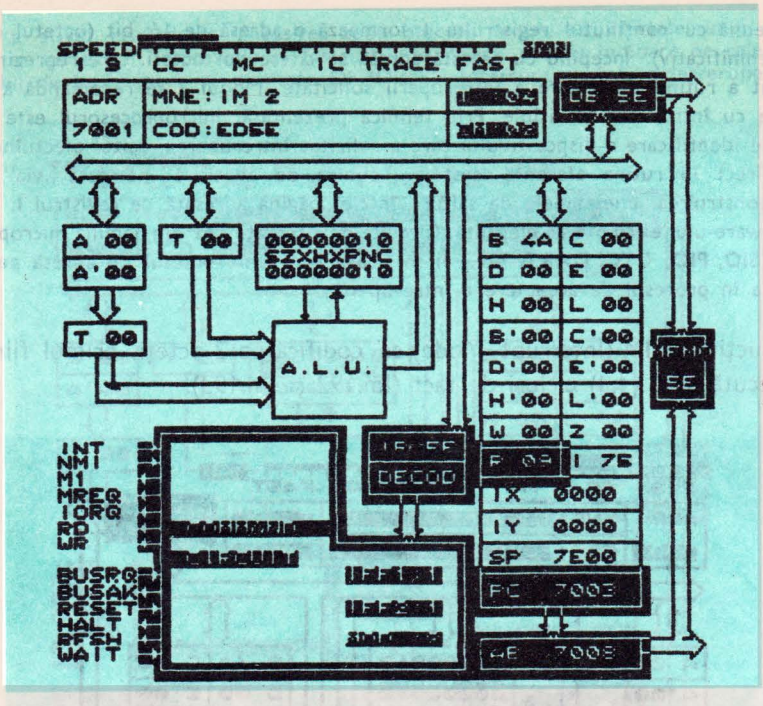
Instrucțiunile **IM** (Interrupt Mode) se codifică pe 2 octeți, primul fiind ED_H, și se execută în 8 (4,4) cicluri de tact (Im.192 și Im.193).



Im.192

(O ultimă secvență din IM 2)

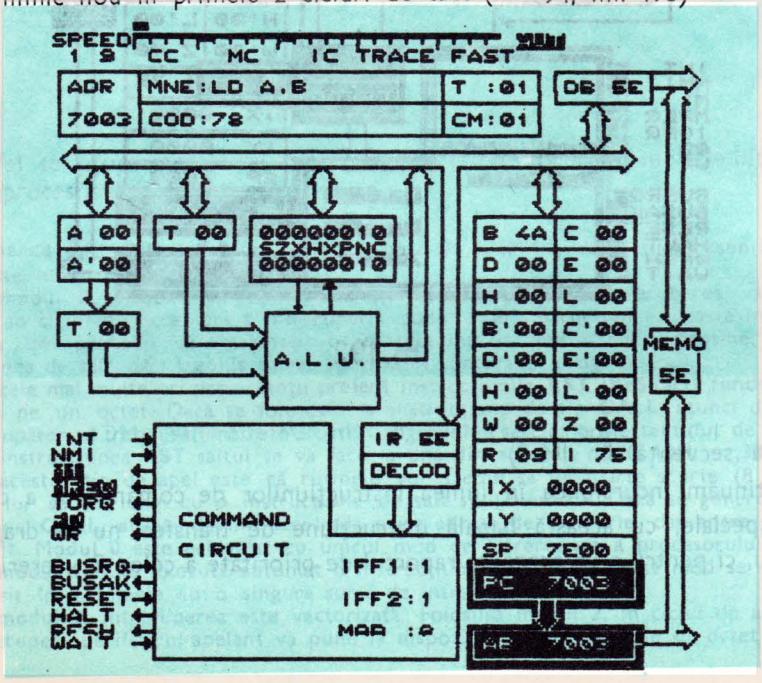
Continuăm incursiunea în lumea instrucțiunilor de comandă și a ciclurilor mașină speciale, cu această banală instrucțiune de transfer, nu de dragul unei diversiuni, ci pentru a vă prezenta raportul de prioritate a celor 2 „cereri" enun-



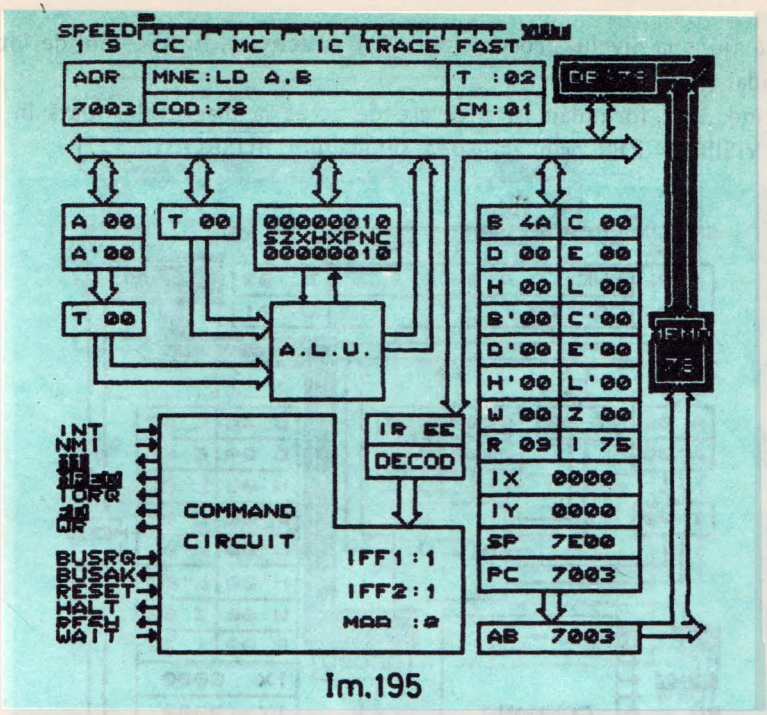
Im. 193

țate în titlul paragrafului : cererea de cedare a magistralelor ($\overline{BUSRQ}$) și cea de întrerupere ($\overline{INT}$).

Așadar nimic nou în primele 2 cicluri de tact (Im.194, Im. 195)

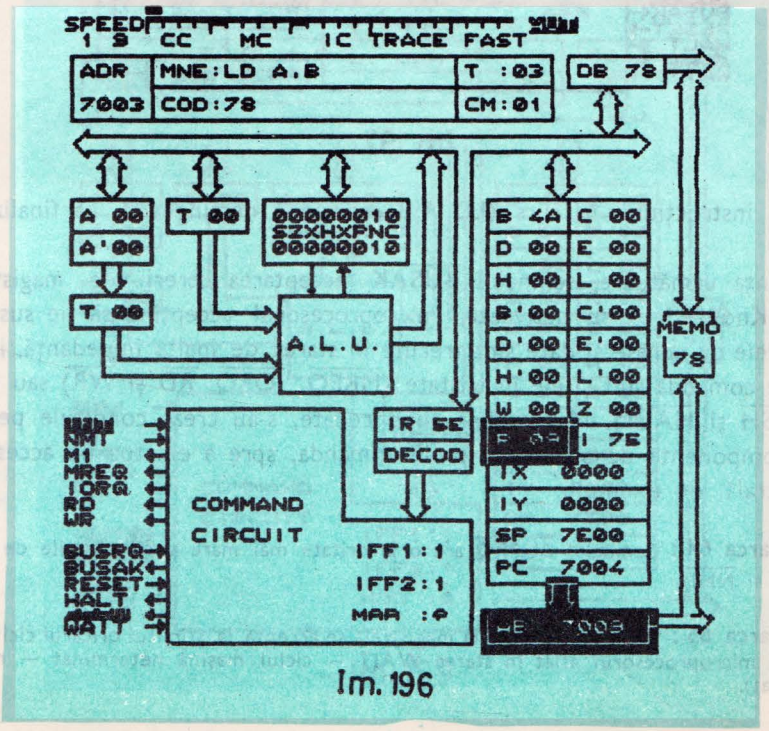


Im. 194



Im.195

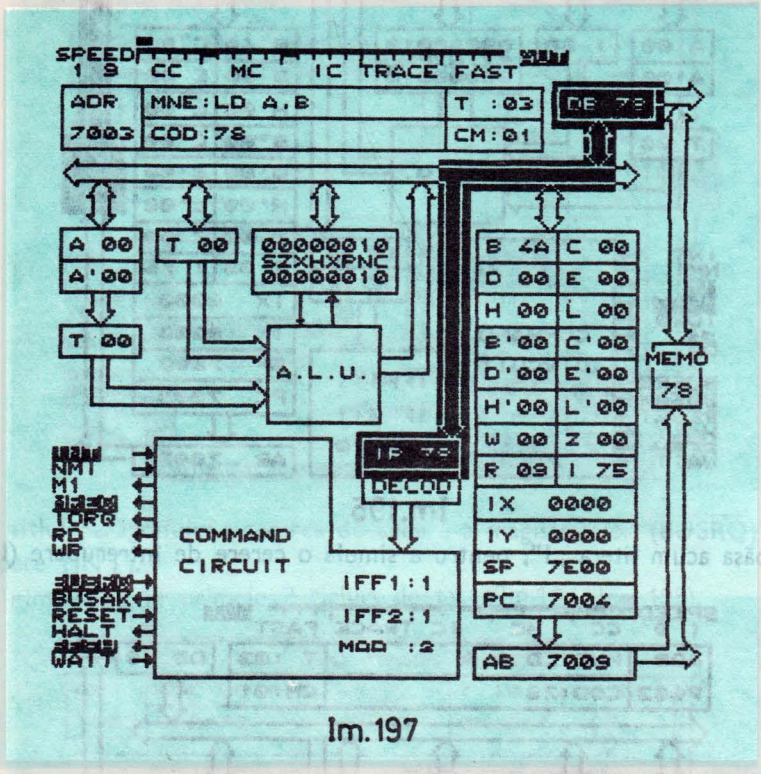
Vom apăsa acum litera „I", pentru a simula o cerere de întrerupere (Im.196).



Im.196

Cererea a ajuns la nivelul procesorului ($\overline{INT}$ — activat), iar sistemul de întreruperi este validat ($IFF_1 = IFF_2 = 1$).

Tastînd „B", formulăm și o cerere de acces la magistrale, luată în considerare de VISIBLE—Z80 prin activarea semnalului $\overline{BUSRQ}$ (Im.197)



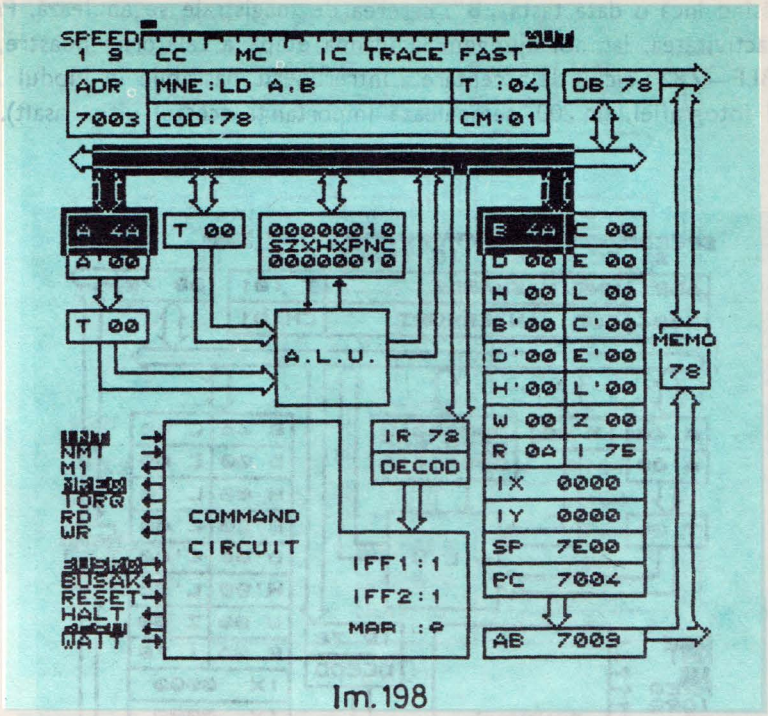
Im.197

Execuția instrucțiunii în curs (LD A,B) continuă nestingherit, ea finalizîndu-se (Im.198).

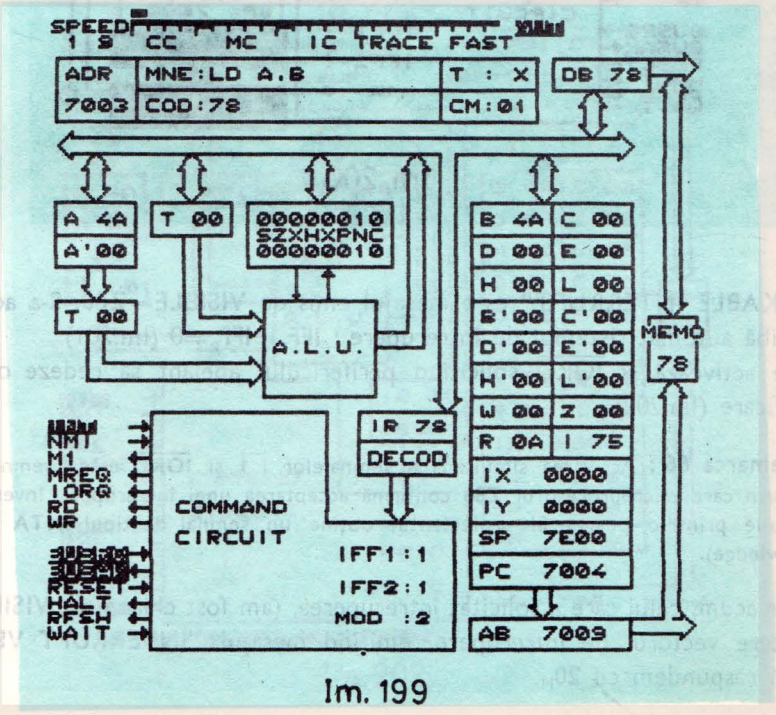
În secvența următoare, semnalul $\overline{BUSAK}$ (acceptarea cererii de magistrale — BUS AcKnowledge) se activează, microprocesorul acceptînd să fie suspendat. Magistralele de adrese și date sînt trecute în starea de înaltă impedanță, iar semnalele de comandă sînt și ele în tristate ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ și $\overline{WR}$) sau inactive ($\overline{M1}$, $\overline{RFSH}$ și $\overline{HALT}$). Magistralele fiind cedate, s-au creat condițiile pentru ca o altă componentă de sistem să preia comanda, spre a efectua un acces direct la magistrală, de exemplu.

Remarca 64 : Semnalul $\overline{BUSRQ}$ are o prioritate mai mare decît cererile de întreruperi ($\overline{INT}$ și $\overline{NMI}$).

Remarca 65 : Cererile de acces la magistrală se acceptă la sfîrșitul oricărui ciclu *mașină* (rezultă că microprocesorul aflat în starea WAIT — ciclul mașină neterminat — nu poate fi suspendat).

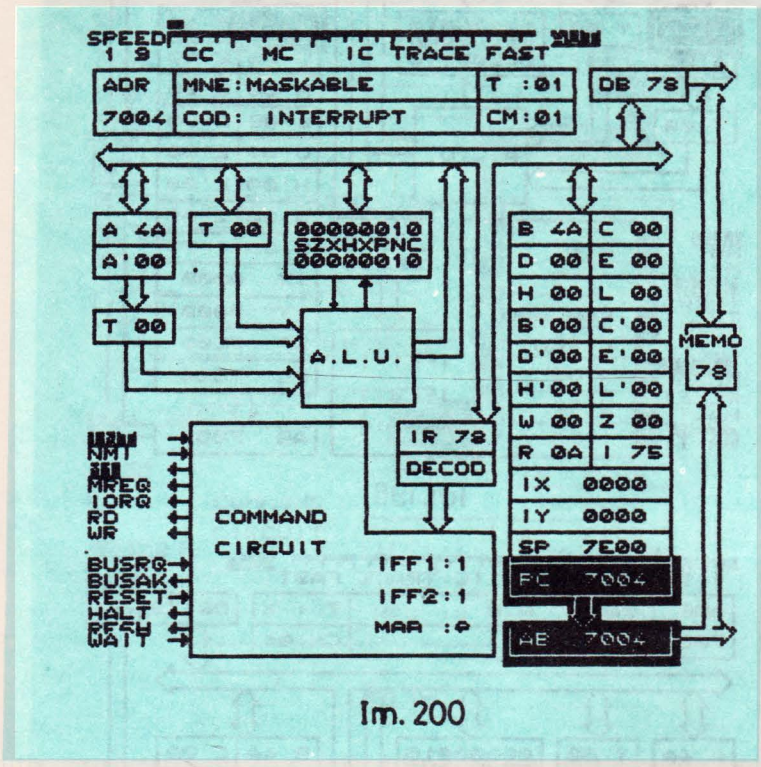

SPEED
1 S CC MC IC TRACE FAST
ADR
7003
MNE:LD A,B
COD:78
T :04
CM:01
DB 78
A 4A
A'00
T 00
00000010
SZXHXPNC
00000010
B 4A
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 0A
I 75
IX 0000
IY 0000
SP 7E00
PC 7004
AB 7009
MEMO
78
T 00
A.L.U.
IR 78
DECOD
NMI
M1
IORQ
RD
WR
BUSAK
RESET
HALT
WAIT
COMMAND
CIRCUIT
IFF1:1
IFF2:1
MAR :0

Im.198


SPEED
1 S CC MC IC TRACE FAST
ADR
7003
MNE:LD A,B
COD:78
T : X
CM:01
DB 78
A 4A
A'00
T 00
00000010
SZXHXPNC
00000010
B 4A
C 00
D 00
E 00
H 00
L 00
B'00
C'00
D'00
E'00
H'00
L'00
W 00
Z 00
R 0A
I 75
IX 0000
IY 0000
SP 7E00
PC 7004
AB 7009
MEMO
78
T 00
A.L.U.
IR 78
DECOD
NMI
M1
MREQ
IORQ
RD
WR
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:1
IFF2:1
MOD :2

Im. 199

Apăsînd încă o dată tasta „B", cererea de magistrale se anulează. Procesorul își reia activitatea, iar noi ajungem la ultima etapă a călătoriei noastre, mediată de VISIBLE—Z80 ; ciclul de acceptare a întreruperii mascabile în Modul 2. (Însuși numărul fotografiei, Im.200, semnalează importanța acestui ultim asalt).



Im. 200

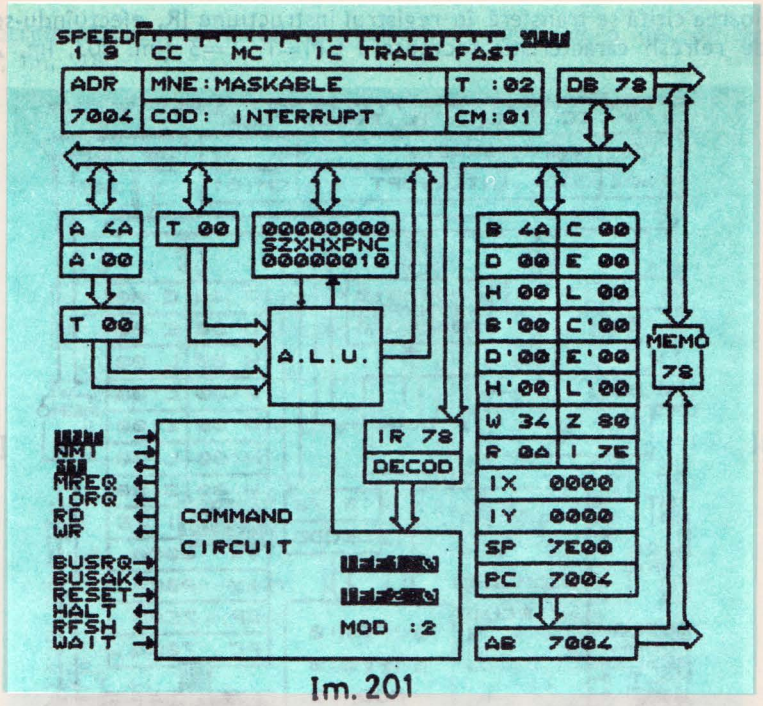
„MASKABLE INTERRUPT" este mesajul emis de VISIBLE—Z80. S-a activat $\overline{M1}$. Se inhibă automat sistemul de întrerupere : $IFF_1 = IFF_2 = 0$ (Im.201).

Se activează și $\overline{IORQ}$ solicitînd perifericului apelant să cedeze octetul de identificare (Im.202).

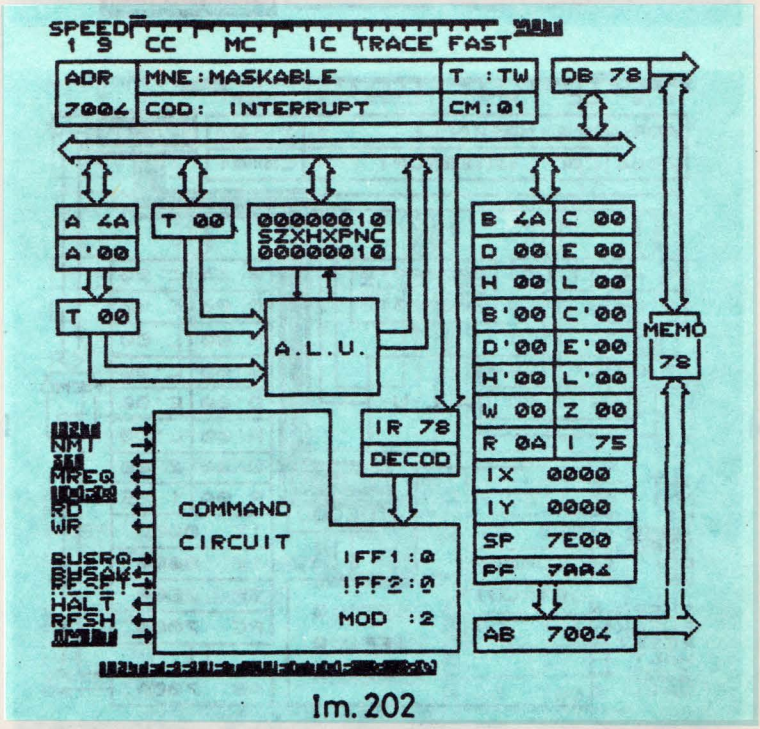
Remarca 66 : Activarea simultană a semnalelor $\overline{M1}$ și $\overline{IORQ}$ este „semnalul" hardware, prin care microprocesorul **Z80** confirmă acceptarea unei întreruperi. Inversîndu-le și trecîndu-le printr-o poartă **SI** proiectantul obține un semnal de tipul **INTA** (INTerrupt Acknowledge).

Iar acum, celui care a solicitat întreruperea, (am fost chiar noi) VISIBLE—Z80 îi va cere vectorul de întrerupere, emițînd mesajul "INTERRUPT VECTOR ?"

Să răspundem cu 20_H.

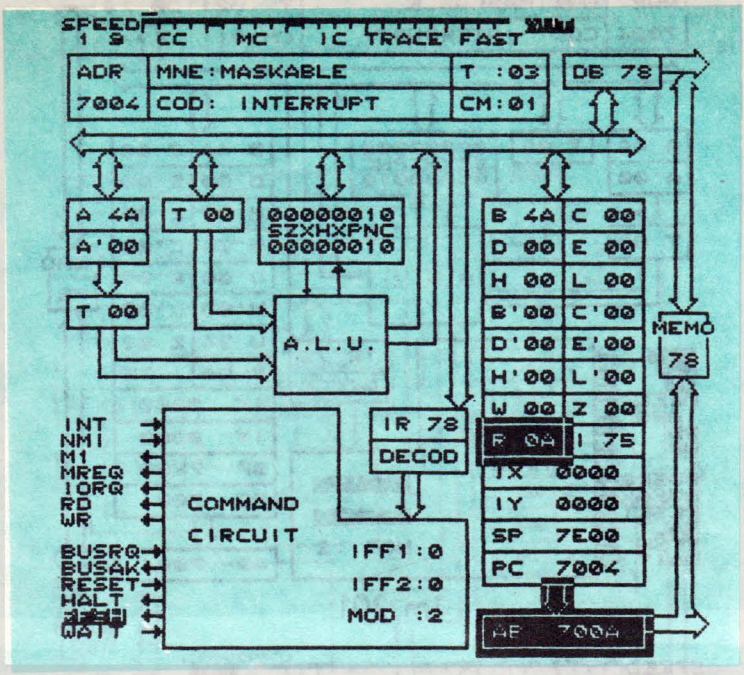

SPEED
1 S CC MC IC TRACE FAST
ADR
7004
MNE:MASKABLE
COD: INTERRUPT
T :02
CM:01
DB 78
A 4A
A'00
T 00
00000000
SZXHXPNC
00000010
T 00
A.L.U.
B 4A C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 34 Z 80
R 0A I 75
IX 0000
IY 0000
SP 7E00
PC 7004
MEMO
78
IR 78
DECOD
NMI
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
MOD :2
AB 7004

Im. 201

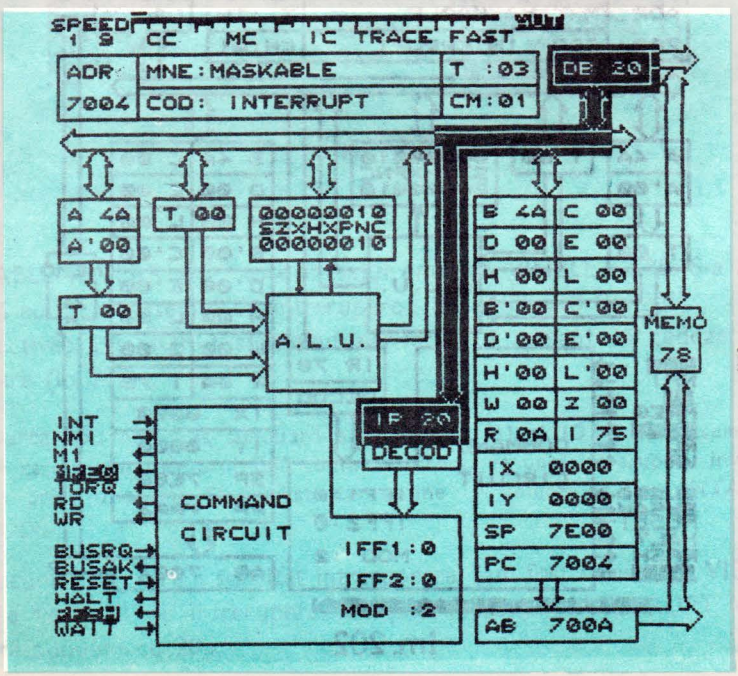

SPEED
1 S CC MC IC TRACE FAST
ADR
7004
MNE:MASKABLE
COD: INTERRUPT
T :TW
CM:01
DB 78
A 4A
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 4A C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 00 Z 00
R 0A I 75
IX 0000
IY 0000
SP 7E00
PC 7004
MEMO
78
IR 78
DECOD
NMI
MREQ
RD
WR
BUSRQ
BUSAK
HALT
RFSH
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :2
AB 7004

Im. 202

Valoarea citită se transferă în registrul instrucțiune **IR**, efectuîndu-se totodată ciclul de refresh caracteristic secvenței : CM=1, T=3 (Im. 203, Im. 204).

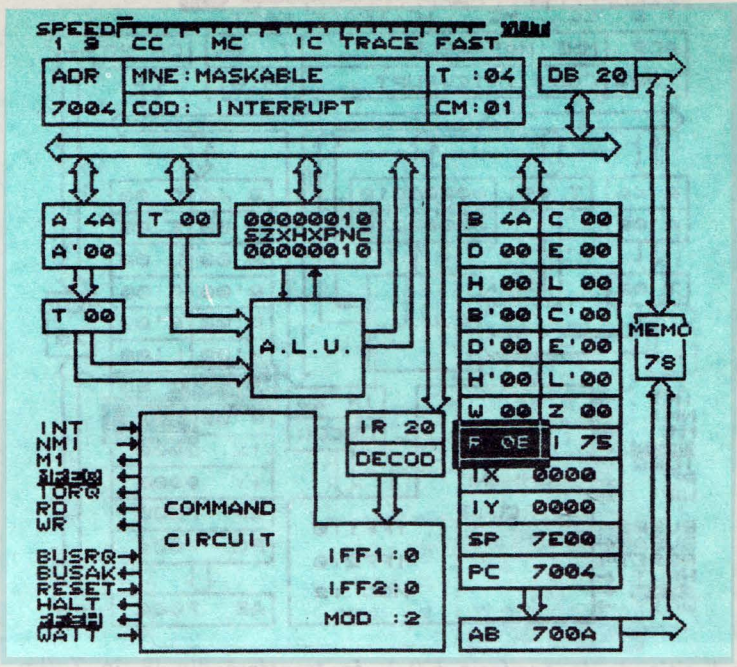


Im. 203

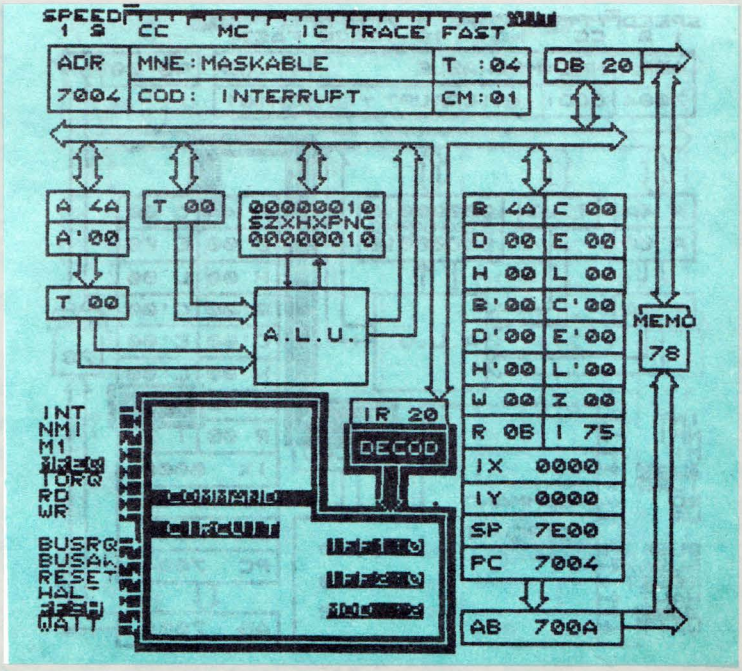


Im. 204

După incrementarea registrului de refresh R, urmează o decodificare falsă : (Im. 205, Im. 206).

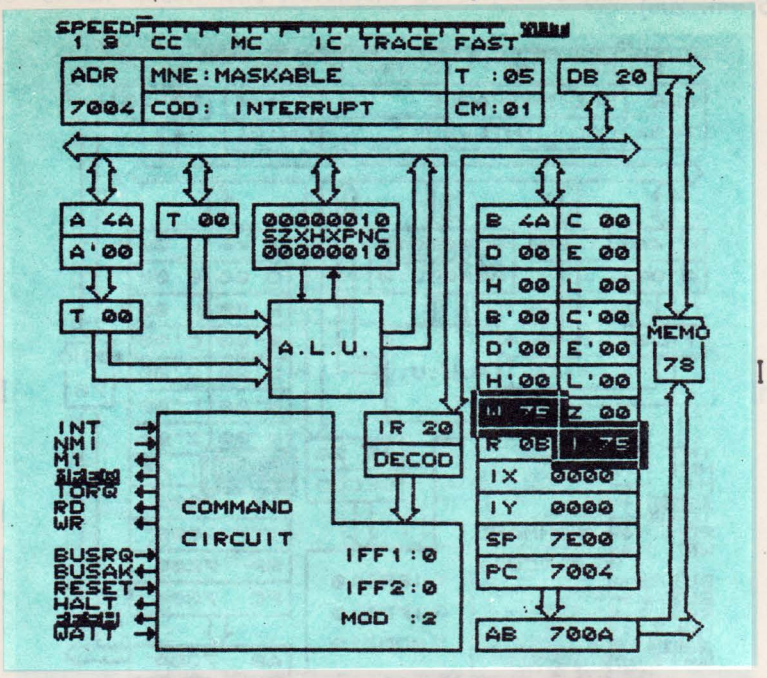


Im. 205



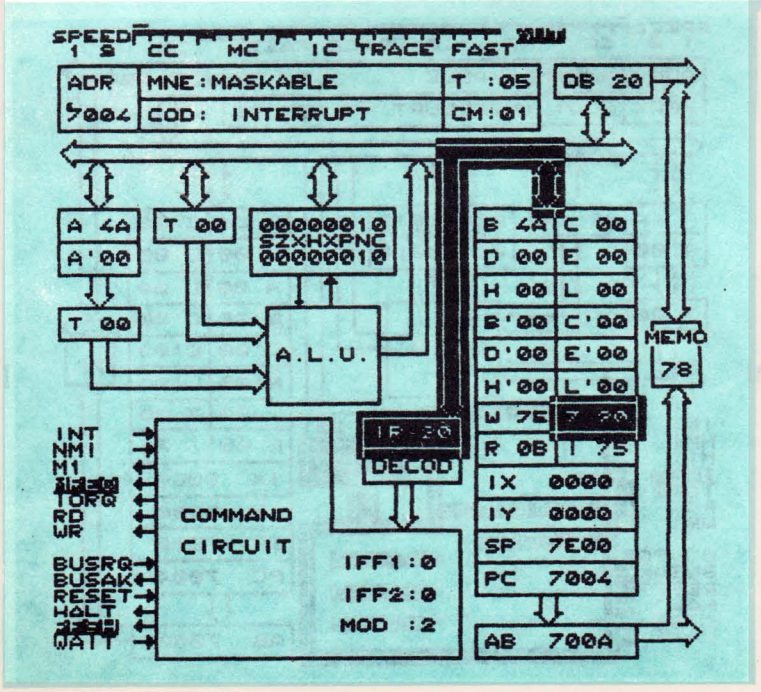
Im. 206

Urmează generarea vectorului complet, pe 16 bit: registrul de întrerupere I se transferă în registrul tampon W (Im. 207),



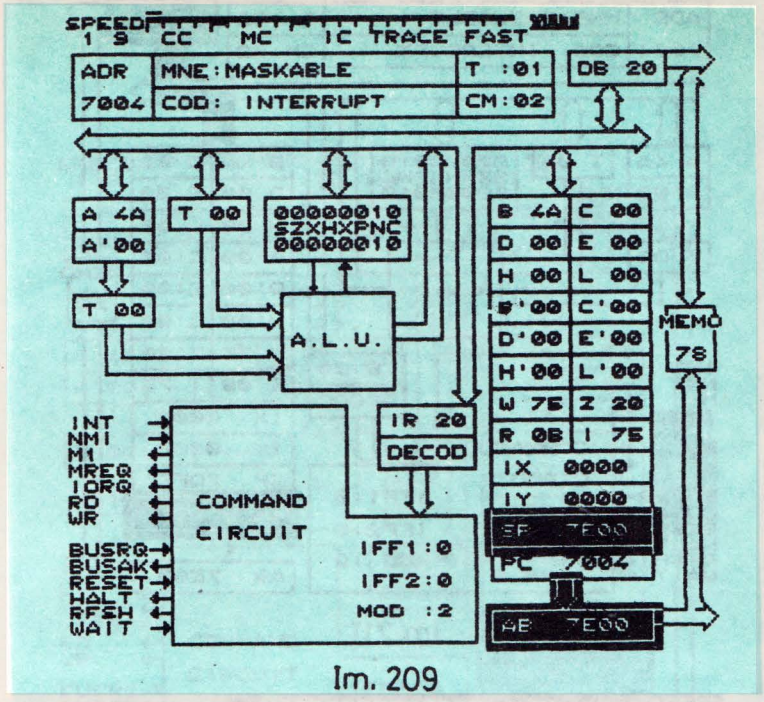
Im. 207

și vectorul citit de la periferic (20_H), se transferă din IR în Z (Im. 208).

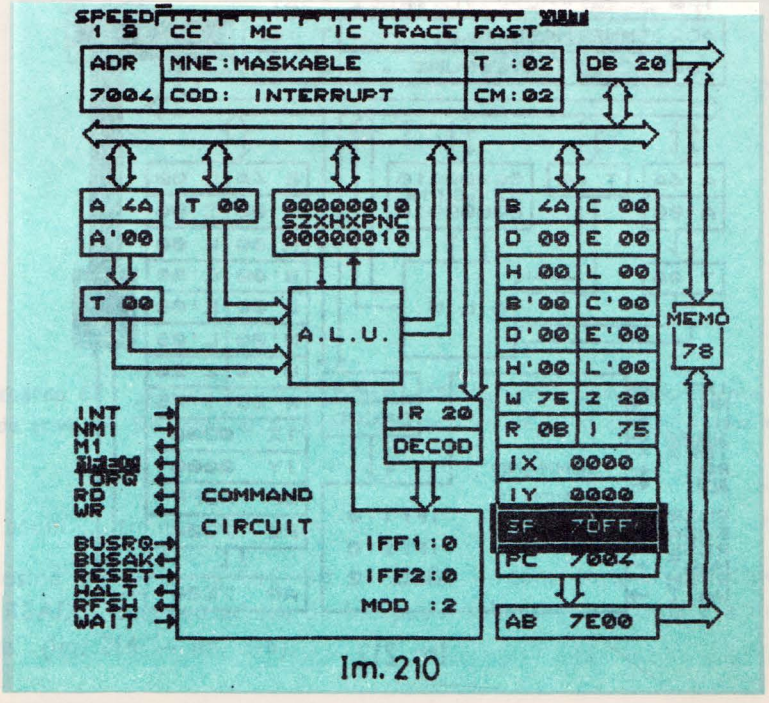


Im. 208

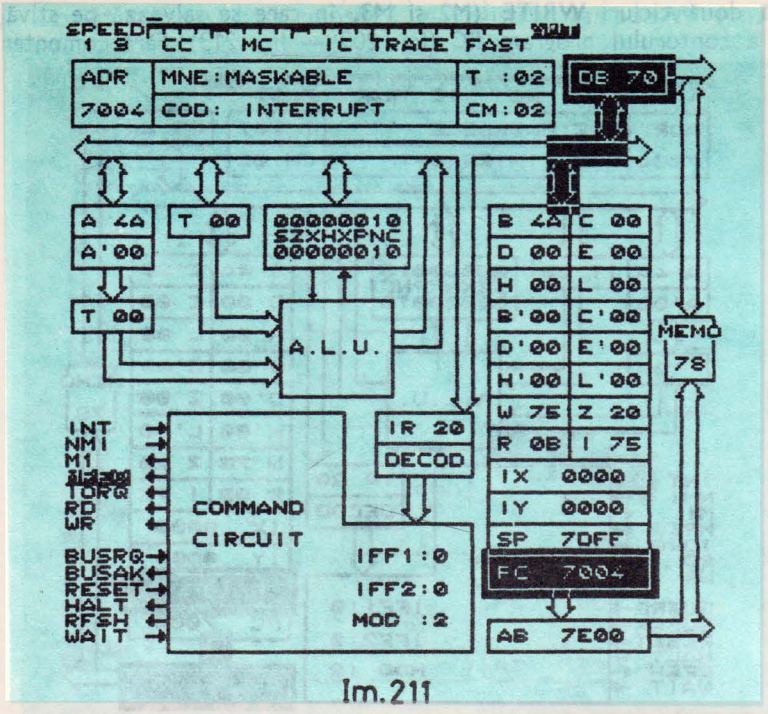
Urmează două cicluri **WRITE** (**M2** și **M3**, în care se salvează pe stivă valoarea curentă a contorului program PC (Im. 209 — Im. 213). Fără comentarii :



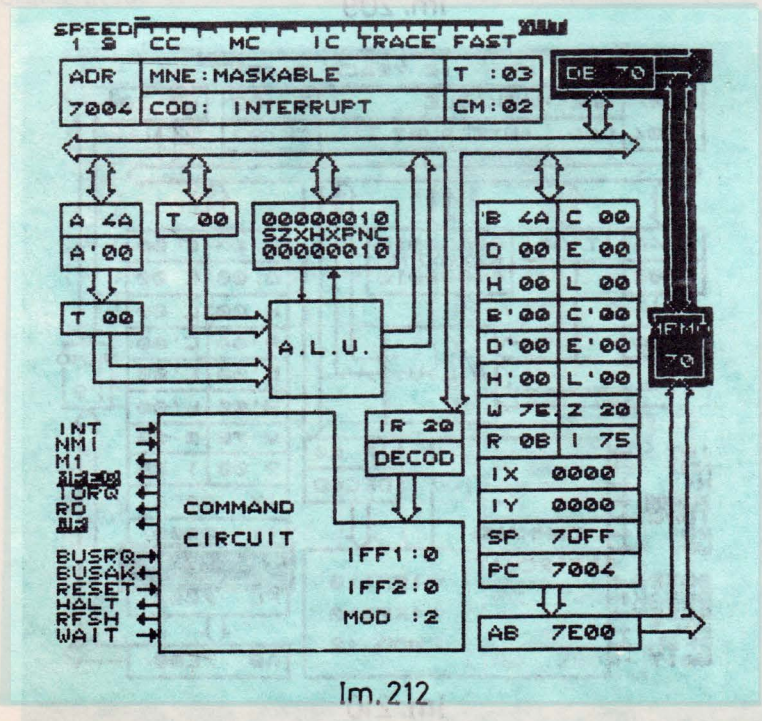
Im. 209



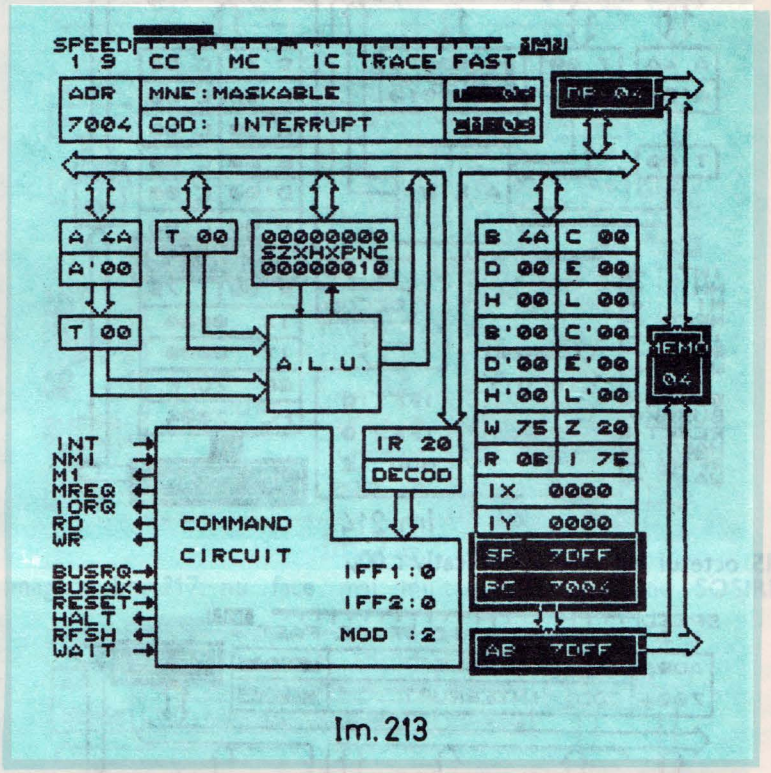
Im. 210


SPEED
1 S CC MC IC TRACE FAST
ADR
7004
MNE:MASKABLE
COD: INTERRUPT
T :02
CM:02
DB 70
A 4A
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 4A C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 75 Z 20
R 0B I 75
IX 0000
IY 0000
SP 7DFF
PC 7004
AB 7E00
MEMO
78
IR 20
DECOD
INT
NMI
M1
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :2

Im.211


SPEED
1 S CC MC IC TRACE FAST
ADR
7004
MNE:MASKABLE
COD: INTERRUPT
T :03
CM:02
DB 70
A 4A
A'00
T 00
00000010
SZXHXPNC
00000010
T 00
A.L.U.
B 4A C 00
D 00 E 00
H 00 L 00
B'00 C'00
D'00 E'00
H'00 L'00
W 75 Z 20
R 0B I 75
IX 0000
IY 0000
SP 7DFF
PC 7004
AB 7E00
MEMO
70
IR 20
DECOD
INT
NMI
M1
IORQ
RD
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :2

Im.212



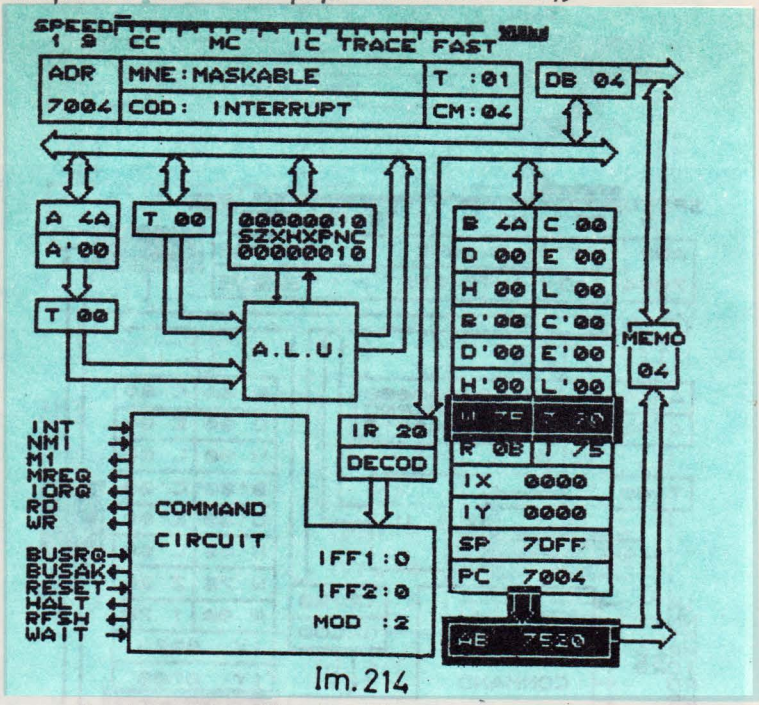
Im. 213

Remarca 67 : „Obosit şi el", **VISIBLE—Z80** a omis prima decrementare a indicatorului de stivă **SP**. De aceea el a salvat **PC**-ul la adresele 7E00 şi 70FF în loc de 7DFF şi 7DFE.

Şi acum „ultimum ultimorum" :

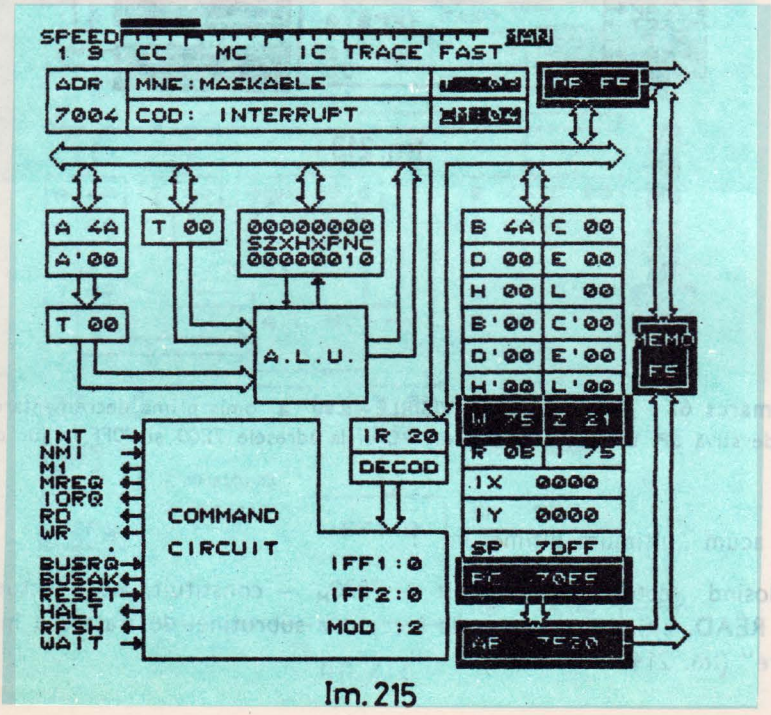
Folosind vectorul de 16 bit — 7520_H — constituit, se efectuează două cicluri **READ**, citindu-se adresa de început a subrutinei de tratare a întreruperii „noastre" (Im. 214 — Im. 216).

În **M4** se citește octetul cel mai puțin semnificativ : $E5_H$



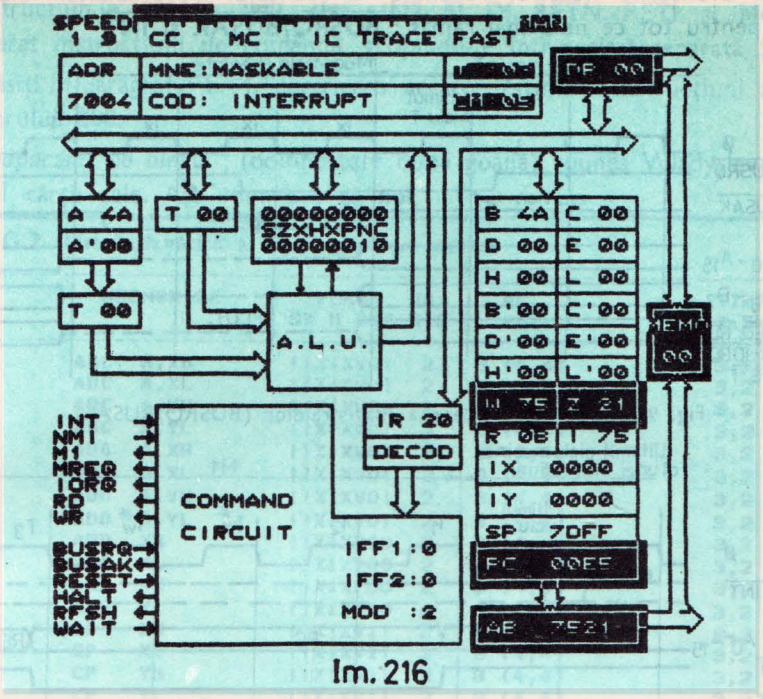
Im. 214

Iar în **M5** octetul cel mai semnificativ : 00_H



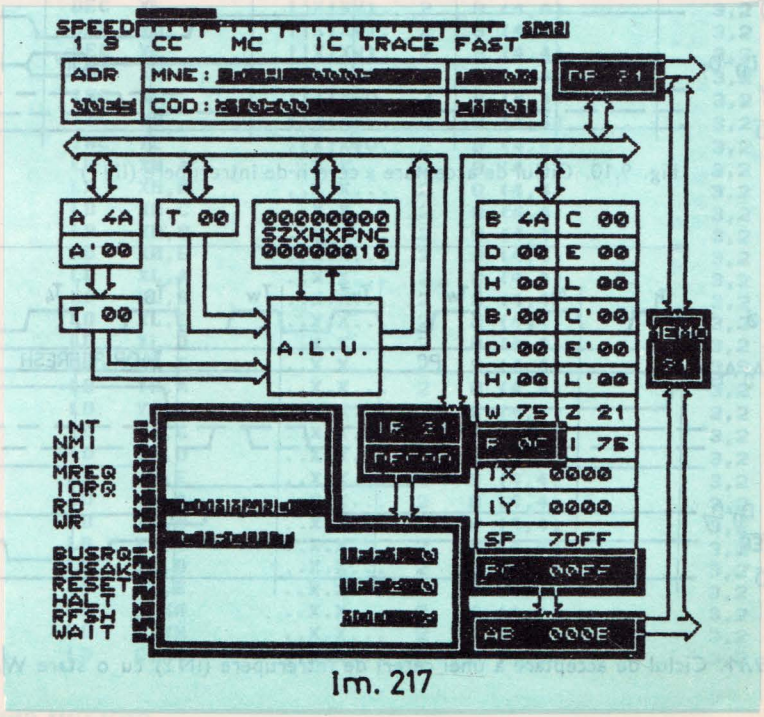
Im. 215

Adresa completă $00E5_H$ se depune în PC și astfel se efectuează saltul dorit.



Im. 216

Ultima imagine Im. 217, nu face mai mult decît să confirme **SOSIREA.**



Im. 217

Terminăm aventurile noastre, revenind la realitate. Vă prezentăm diagramele de timp pentru tot ce ne-a mai rămas : **BUSRQ/BUSAK** și **INT**

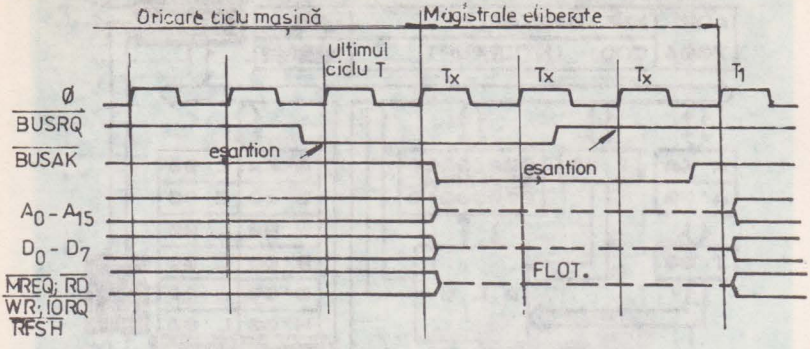


Fig. 9.9. Ciclul de cedare a magistralelor (BUSRQ/BUSAK)

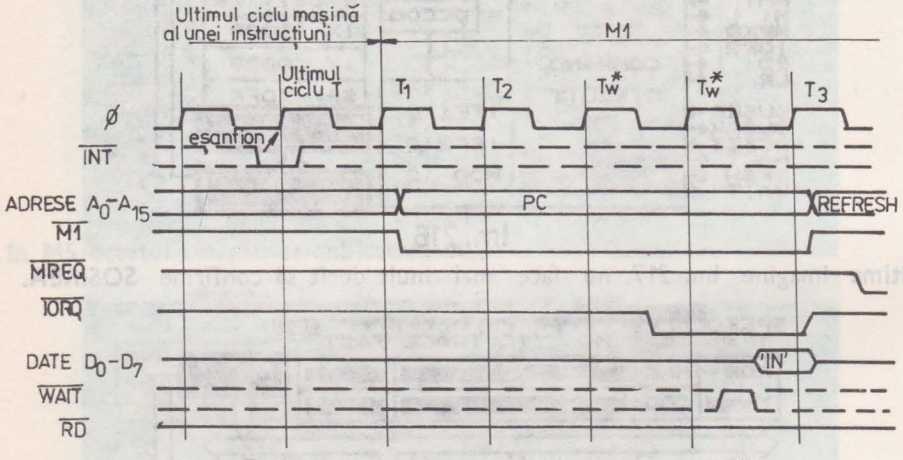


Fig. 9.10. Ciclul de acceptare a cererii de întrerupere (INT)

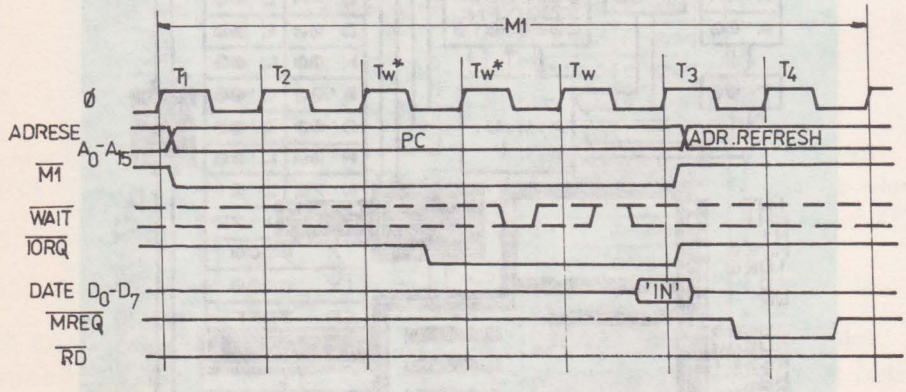


Fig. 9.11. Ciclul de acceptare a unei cereri de întrerupere (INT) cu o stare WAIT

9.5. Sinteza clasei : SYS

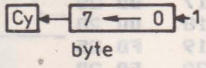
Instrucţiunile **HALT, IM0, IM1, IM2, EI, DI, RETN, RETI** şi **NOP** le-am considerat instrucţiuni de comandă, grupîndu-le într-o clasă separată **SYS.**

Găsiţi în paragraful E.14. descrierea acestor ultime 9 instrucţiuni ale microprocesorului **Z80.**

"Odpacznij po biegu" (odihneşte-te după goană) spunea Wladyslaw Terlecki în titlul cărţii sale. Aşa vom face şi noi.

tabela **G.2** **Lista alfabetică a instrucţiunilor ascunse**

NR. CRT	COD	MNEMONICA	FLAG SZ H PNC	NR. CIC	NR. TACTI	TIMP 2.5M	COMENTARII
1	DD 8C	ADC A,XH	!!X!XVO!	2	8 (4,4)	3,2	
2	DD 8D	ADC A,XL	!!X!XVO!	2	8 (4,4)	3,2	
3	FD 8C	ADC A,YH	!!X!XVO!	2	8 (4,4)	3,2	
4	FD 8D	ADC A,YL	!!X!XVO!	2	8 (4,4)	3,2	
5	DD 84	ADD A,XH	!!X!XVO!	2	8 (4,4)	3,2	
6	DD 85	ADD A,XL	!!X!XVO!	2	8 (4,4)	3,2	
7	FD 84	ADD A,YH	!!X!XVO!	2	8 (4,4)	3,2	
8	FD 85	ADD A,YL	!!X!XVO!	2	8 (4,4)	3,2	
9	DD A4	AND XH	!!X1XPOO	2	8 (4,4)	3,2	
10	DD A5	AND XL	!!X1XPOO	2	8 (4,4)	3,2	
11	FD A4	AND YH	!!X1XPOO	2	8 (4,4)	3,2	
12	FD A5	AND YL	!!X1XPOO	2	8 (4,4)	3,2	
13	DD BC	CP XH	!!X!XV1!	2	8 (4,4)	3,2	
14	DD BD	CP XL	!!X!XV1!	2	8 (4,4)	3,2	
15	FD BC	CP YH	!!X!XV1!	2	8 (4,4)	3,2	
16	FD BD	CP YL	!!X!XV1!	2	8 (4,4)	3,2	
17	DD 25	DEC XH	!!X!XV1.	2	8 (4,4)	3,2	
18	DD 2D	DEC XL	!!X!XV1.	2	8 (4,4)	3,2	
19	FD 25	DEC YH	!!X!XV1.	2	8 (4,4)	3,2	
20	FD 2D	DEC YL	!!X!XV1.	2	8 (4,4)	3,2	
21	FD 24	INC XH	!!X!XVO.	2	8 (4,4)	3,2	
22	DD 2C	INC XL	!!X!XVO.	2	8 (4,4)	3,2	
23	FD 24	INC YH	!!X!XVO.	2	8 (4,4)	3,2	
24	FD 2C	INC YL	!!X!XVO.	2	8 (4,4)	3,2	
25	DD 67	LD XH,A	..X.X...	2	8 (4,4)	3,2	
26	DD 60	LD XH,B	..X.X...	2	8 (4,4)	3,2	
27	DD 61	LD XH,C	..X.X...	2	8 (4,4)	3,2	
28	DD 62	LD XH,D	..X.X...	2	8 (4,4)	3,2	
29	DD 63	LD XH,E	..X.X...	2	8 (4,4)	3,2	
30	DD 6F	LD XL,A	..X.X...	2	8 (4,4)	3,2	
31	DD 68	LD XL,B	..X.X...	2	8 (4,4)	3,2	
32	DD 69	LD XL,C	..X.X...	2	8 (4,4)	3,2	
33	DD 6A	LD XL,D	..X.X...	2	8 (4,4)	3,2	
34	DD 6B	LD XL,E	..X.X...	2	8 (4,4)	3,2	
35	FD 67	LD YH,A	..X.X...	2	8 (4,4)	3,2	
36	FD 60	LD YH,B	..X.X...	2	8 (4,4)	3,2	
37	FD 61	LD YH,C	..X.X...	2	8 (4,4)	3,2	
38	FD 62	LD YH,D	..X.X...	2	8 (4,4)	3,2	
39	FD 63	LD YH,E	..X.X...	2	8 (4,4)	3,2	
40	FD 6F	LD YL,A	..X.X...	2	8 (4,4)	3,2	
41	FD 68	LD YL,B	..X.X...	2	8 (4,4)	3,2	
42	FD 69	LD YL,C	..X.X...	2	8 (4,4)	3,2	
43	FD 6A	LD YL,D	..X.X...	2	8 (4,4)	3,2	
44	FD 6B	LD YL,E	..X.X...	2	8 (4,4)	3,2	
45	DD 7C	LD A,XH	..X.X...	2	8 (4,4)	3,2	
46	DD 44	LD B,XH	..X.X...	2	8 (4,4)	3,2	
47	DD 4C	LD C,XH	..X.X...	2	8 (4,4)	3,2	

NR. CRT	COD	MNEMONICA		FLAG SZ H PNC	NR. CIC	NR. TACTI	TIMP 2.5M	COMENTARII
48	DD 54	LD	D,XH	..X.X...	2	8 (4,4)	3,2	
49	DD 5C	LD	E,XH	..X.X...	2	8 (4,4)	3,2	
50	DD 7D	LD	A,XL	..X.X...	2	8 (4,4)	3,2	
51	DD 45	LD	B,XL	..X.X...	2	8 (4,4)	3,2	
52	DD 4D	LD	C,XL	..X.X...	2	8 (4,4)	3,2	
53	DD 55	LD	D,XL	..X.X...	2	8 (4,4)	3,2	
54	DD 5D	LD	E,XL	..X.X...	2	8 (4,4)	3,2	
55	FD 7C	LD	A,YH	..X.X...	2	8 (4,4)	3,2	
56	FD 44	LD	B,YH	..X.X...	2	8 (4,4)	3,2	
57	FD 4C	LD	C,YH	..X.X...	2	8 (4,4)	3,2	
58	FD 54	LD	D,YH	..X.X...	2	8 (4,4)	3,2	
59	FD 5C	LD	E,YH	..X.X...	2	8 (4,4)	3,2	
60	FD 7D	LD	A,YL	..X.X...	2	8 (4,4)	3,2	
61	FD 45	LD	B,YL	..X.X...	2	8 (4,4)	3,2	
62	FD 4D	LD	C,YL	..X.X...	2	8 (4,4)	3,2	
63	FD 55	LD	D,YL	..X.X...	2	8 (4,4)	3,2	
64	FD 5D	LD	E,YL	..X.X...	2	8 (4,4)	3,2	
65	DD 26 20	LD	XH,N	..X.X...	3	11 (4,4,3)	4,4	
66	DD 2E 20	LD	XL,N	..X.X...	3	11 (4,4,3)	4,4	
67	FD 26 20	LD	YH,N	..X.X...	3	11 (4,4,3)	4,4	
68	FD 2E 20	LD	YL,N	..X.X...	3	11 (4,4,3)	4,4	
69	DD B4	OR	XH	!!XOXP00	2	8 (4,4)	3,2	
70	DD B5	OR	XL	!!XOXP00	2	8 (4,4)	3,2	
71	FD B4	OR	YH	!!XOXP00	2	8 (4,4)	3,2	
72	FD B5	OR	YL	!!XOXP00	2	8 (4,4)	3,2	
73	DD 9C	SBC	XH,A	!!X!XV1!	2	8 (4,4)	3,2	
74	DD 9D	SBC	XL,A	!!X!XV1!	2	8 (4,4)	3,2	
75	FD 9C	SBC	YH,A	!!X!XV1!	2	8 (4,4)	3,2	
76	FD 9D	SBC	YL,A	!!X!XV1!	2	8 (4,4)	3,2	
77	CB 37	SLH	A	!!XOXPO!	2	8 (4,4)	3,2	
78	CB 30	SLH	B	!!XOXPO!	2	8 (4,4)	3,2	
79	CB 31	SLH	C	!!XOXPO!	2	8 (4,4)	3,2	
80	CB 32	SLH	D	!!XOXPO!	2	8 (4,4)	3,2	
81	CB 33	SLH	E	!!XOXPO!	2	8 (4,4)	3,2	
82	CB 34	SLH	H	!!XOXPO!	2	8 (4,4)	3,2	
83	CB 35	SLH	L	!!XOXPO!	2	8 (4,4)	3,2	
84	CB 36	SLH	(HL)	!!XOXPO!	4	15 (4,4,4,3)	6,0	
85	DD CB 05 36	SLH	(IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9,2	
86	FD CB 05 36	SLH	(IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9,2	
87	DD 94	SUB	XH	!!X!XV1!	2	8 (4,4)	3,2	
88	DD 95	SUB	XL	!!X!XV1!	2	8 (4,4)	3,2	
89	FD 94	SUB	YH	!!X!XV1!	2	8 (4,4)	3,2	
90	FD 95	SUB	YL	!!X!XV1!	2	8 (4,4)	3,2	
91	DD AC	XOR	XH	!!XOXP00	2	8 (4,4)	3,2	
92	DD AD	XOR	XL	!!XOXP00	2	8 (4,4)	3,2	
93	FD AC	XOR	YH	!!XOXP00	2	8 (4,4)	3,2	
94	FD AD	XOR	YL	!!XOXP00	2	8 (4,4)	3,2	

Partea a II-a

COMPLEMENTE PRIVIND MICROPROCESORUL Z80 ŞI SIMULATORUL SĂU

HEX.	COD			MNEMONICA	FLAG	BR.	BR.	TIMP	COMENTARII
CRT					SS R PRC	CIC	TACTI	2.5M	
48	00 54			LD B,XH	..X.X...	2	8 (4,4)	3,2	
49	00 5C			LD E,XH	..X.X...	2	8 (4,4)	3,2	
4A	00 78			LD A,XL	..X.X...	2	8 (4,4)	3,2	
4B	00 45			LD B,XL	..X.X...	2	8 (4,4)	3,2	
4C	00 4B			LD C,XL	..X.X...	2	8 (4,4)	3,2	
4D	00 53			LD D,XL	..X.X...	2	8 (4,4)	3,2	
4E	00 5D			LD E,XL	..X.X...	2	8 (4,4)	3,2	
4F	FD 7C			LD A,YH	..X.X...	2	8 (4,4)	3,2	
50	FD 44			LD B,YH	..X.X...	2	8 (4,4)	3,2	
51	FD 4C			LD C,YH	..X.X...	2	8 (4,4)	3,2	
52	FD 54			LD D,YH	..X.X...	2	8 (4,4)	3,2	
53	FD 5C			LD E,YH	..X.X...	2	8 (4,4)	3,2	
54	FD 7D			LD A,YL	..X.X...	2	8 (4,4)	3,2	
55	FD 45			LD B,YL	..X.X...	2	8 (4,4)	3,2	
56	FD 4D			LD C,YL	..X.X...	2	8 (4,4)	3,2	
57	FD 55			LD D,YL	..X.X...	2	8 (4,4)	3,2	
58	FD 5D			LD E,YL	..X.X...	2	8 (4,4)	3,2	
59	FD 26 20			LD XH,N	..X.X...	3	11 (4,4,3)	4,4	
5A	FD 2E 20			LD XL,N	..X.X...	3	11 (4,4,3)	4,4	
5B	FD 26 20			LD YH,N	..X.X...	3	11 (4,4,3)	4,4	
5C	FD 2E 20			LD YL,N	..X.X...	3	11 (4,4,3)	4,4	
5D	DD 84			OR XH	?!XOXPOO	2	8 (4,4)	3,2	
5E	DD 85			OR XL	?!XOXPOO	2	8 (4,4)	3,2	
5F	FD 84			OR YH	?!XOXPOO	2	8 (4,4)	3,2	
60	FD 85			OR YL	?!XOXPOO	2	8 (4,4)	3,2	
61	DD 9C			SBC XH,A	?!X?XV??	2	8 (4,4)	3,2	
62	DD 9D			SBC XL,A	?!X?XV??	2	8 (4,4)	3,2	
63	FD 9C			SBC YH,A	?!X?XV??	2	8 (4,4)	3,2	
64	FD 9D			SBC YL,A	?!X?XV??	2	8 (4,4)	3,2	
65	CB 27			SLA A	?!XOXPO?	2	8 (4,4)	3,2	
66	CB 35			SLA L	?!XOXPO?	2	8 (4,4)	3,2	
67	CB 21			SLA C	?!XOXPO?	2	8 (4,4)	3,2	
68	CB 32			SLA D	?!XOXPO?	2	8 (4,4)	3,2	
69	CB 33			SLA E	?!XOXPO?	2	8 (4,4)	3,2	
6A	CB 34			SLA H	?!XOXPO?	2	8 (4,4)	3,2	
6B	CB 35			SLA L	?!XOXPO?	2	8 (4,4)	3,2	
6C	CB 36			SLA (HL)	?!XOXPO?	4	15 (4,4,4,3)	4,6	
6D	DD CB 05 36			SLA (IX+IMM)	?!XOXPO?	6	23 (4,4,3,5,4,3)	5,2	
6E	FD CB 05 36			SLA (IY+IMM)	?!YOXPO?	6	23 (4,4,3,5,4,3)	5,2	
6F	DD 94			SBC XH	?!?!XV??	2	8 (4,4)	3,2	
70	DD 95			SUB XL	?!X?XV??	2	8 (4,4)	3,2	
71	FD 94			SUB YH	?!X?XV??	2	8 (4,4)	3,2	
72	FD 95			SUB YL	?!X?XV??	2	8 (4,4)	3,2	
73	DD AC			XOR XH	?!XOXPOO	2	8 (4,4)	3,2	
74	DD AD			XOR XL	?!XOXPOO	2	8 (4,4)	3,2	
75	FD AC			XOR YH	?!XOXPOO	2	8 (4,4)	3,2	
76	FD AD			XOR YL	?!XOXPOO	2	8 (4,4)	3,2	

A

MANUALUL DE UTILIZARE A CASETEI CU PRODUSUL—PROGRAM VISIBLE—Z80 ŞI CU PROGRAMUL PROIECTULUI CASEI DE MARCAT

Ataşarea unei casete program la o carte este o premieră în practica editorială din România. Scopul care se urmăreşte prin acest experiment este acela, de a pune la dispoziţia cititorului un instrument care poate fi deosebit de util în asimilarea cunoştinţelor despre microprocesorul **Z80.** Utilizînd un calculator personal, această casetă magnetică vă va permite să efectuaţi sesiuni de autoinstruire asistată de calculator, aprofundînd astfel materialul de studiu inclus în prezenta carte.

Caseta conţine programul **VISZ80 (visible microprocessor Z80)** elaborat de către autor, animat de dorinţa de a realiza un instrument care să permită concretizarea fenomenelor interne intime ale microprocesorului, dilatîndu-le în timp şi spaţiu astfel încît ele să poată fi percepute de sistemul senzorial al omului.

S-a realizat astfel un complex audio-vizual în care mişcarea şi semnalele sonore converg spre un obiectiv comun : asistarea studiului.

VISZ80 este un modul software de 24 kocteţi realizat în întregime în limbaj de asamblare. El se constituie din 6 blocuri funcţionale majore :

— *monitorul propriu* — oferă un set de comenzi care permit exploatarea resurselor **VISZ80.**

— *asamblorul direct* — permite elaborarea unor programe în limbaj de asamblare, încărcînd codul translatat direct în memorie.

— *dezasamblorul* — efectuează transpunerea inversă a programului din cod maşină în mnemonicele limbajului de asamblare ; programele aflate în memorie se traduc şi se listează pe ecran.

— *procesorul grafic* — este un pachet de programe care permit o tratare unitară a tuturor manevrelor ecran invocate de către simulatorul propriu-zis ;

— *simulatorul de instrucţiuni* — simulează (pe cale software) execuţia tuturor instrucţiunilor publicate ale microprocesorului Z80, păstrînd permanent, într-o zonă de memorie proprie, oglinda tuturor regiştrilor interni ai microprocesorului.

— *supervizorul* — dirijează execuţia simulată a programelor cod-maşină, stabilind regimul de lucru şi viteza de simulare a instrucţiunilor ; el rezolvă şi asortarea sonoră selectivă a paşilor de simulare.

În înregistrarea realizată pe caseta **VISIBLE—Z80** au fost înglobate şi cele 67 rutine elaborate în studiul de caz din partea a treia a cărţii (listing cap. 17), ele putînd fi oricînd străbătute cu ajutorul comenzilor oferite de către ambianţa de lucru **VISIBLE—Z80.**

Ataşate, cele 2 module software formează o înregistrare de 28 kocteţi.

VISIBLE Z—80 a fost elaborat mai întîi pentru calculatorul PRAE—M

Programele incluse se găsesc pe prima faţă a casetei. Pentru a veni în sprijinul unui număr cît mai mare de utilizatori posibili, autorii au elaborat şi o implementare a acestui pachet de programe pentru calculatorul personal **aMIC,** înregistrare care se regăseşte pe faţa a doua a casetei.

A.1. Încărcarea şi lansarea programului

Procedeul de încărcare a celor două versiuni de implementare de pe cele 2 feţe ale casetei diferă datorită diferenţelor constructive şi structurale ale celor 2 calculatoare **PRAE** şi **aMIC.** În ambele cazuri caseta se va poziţiona în casetofon pe porţiunea de bandă ştearsă care precede înregistrarea de date.

A.1.1. Comenzi pentru calculatorul PRAE—M

După instalarea BASIC-ului **(V3.5)** marcată prin afişarea mesajului **READY** se lansează comanda de încărcare. Dacă vă aflaţi la primul contact cu caseta **VISIBLE—Z80** vă recomandăm să lansaţi o comandă de probă :

ALOAD " "

Porniţi casetofonul pe poziţia REDARE (PLAY). După trecerea primului bloc de date, în colţul stîng jos al ecranului va trebui să apară titlul înregistrării : **VISIBLE—Z80.** Dacă evenimentul aşteptat se produce, opriţi derularea benzii, apăsaţi butonul **RESET** de pe panoul dorsal al calculatorului şi tastaţi litera **R** pentru a reveni în **BASIC PRAE.**

În acest caz există premisele pentru încărcarea fără eroare a înregistrării de pe casetă în memoria calculatorului. :

Repoziţionaţi caseta în amonte de înregistrarea de date şi lansaţi apoi comanda efectivă de încărcare :

LOAD "VISIBLE—Z80" & 7000

Porniţi casetofonul pe poziţia REDARE (PLAY).

Va trebui să aşteptaţi aproximativ **1 minut 55 secunde** pînă cînd cei 28 kbyte ai înregistrării vor defila în faţa capului de citire a casetofonului.

Odată cu terminarea blocurilor de date, pe ecranul televizorului va trebui să apară mesajul de terminare a încărcării:

— **READY** în cazul unei încărcări fără eroare

— **TAPE ERROR,** urmat de **READY** în cazul unei încărcări cu eroare.

Dacă comanda de probă (**ALOAD " "**) nu a avut efectul scontat, distingem două cazuri :

— pe ecran nu se afișează nimic

— pe ecran se afișează titlul incorect : caractere lipsă sau în plus, eventual se șterge întregul conținut al ecranului.

În primul caz există probabilitatea racordului imperfect între casetofon și calculator. În acest caz va trebui să eliminați lipsa de contact făcînd încercări pe o casetă cuprinzînd înregistrările Dvs., pe care le-ați citit deja.

În cel de-al doilea caz racordul casetofon-calculator poate fi corect, dar semnalul citit de pe bandă nu este bun : el este fie afectat de zgomotul electric din mediu sau din rețea, fie capul de redare nu este reglat la nivelul etalon.

Casetofoanele muzicale nefiind echipate pentru tehnica de calcul sînt puternic afectate de zgomotul electromagnetic. Astfel circuitele de baleiaj și cel de înaltă tensiune ale televizorului, precum și cele ale sursei în comutație a calculatorului pot afecta, prin cîmpul lor electromagnetic calitatea transferului de date. De aceea vă rugam să dispuneți cît mai departe casetofonul de televizor și de sursă, amplasîndu-l cît mai la dreapta calculatorului și atelevizorului. Verificați traseul cablului de redare, ca acesta să nu formeze bucle, care pot acționa ca antene de recepție, și în nici un caz să nu încercuiască televizorul.

Dacă nici aceste măsuri nu aduc rezultatul scontat, ștergeți capetele de înregistrare-redare a casetofonului cu o cîrpă moale îmbibată cu alcool tehnic. Verificați ca banda să se deruleze ușor în casetă. Datorită unor imperfecțiuni ale rolei de antrenare, banda se poate depune neuniform pe rolă, ceea ce îngreunează antrenarea, provocînd variații de viteză la redare. În acest caz vă recomandăm să bateți caseta (pe ambele fețe) de palmă (în dreptul ambelor role) pînă cînd veți constata că ea poate fi derulată ușor (folosind un obiect cu vîrf ex. : pix) în ambele sensuri.

Dacă nici această manevră nu va îmbunătăți transferul de date, atunci probabil capul de redare este decalibrat. Înainte de a vă năpusti cu șurubelnița la șurubul de reglaj, trebuie să fiți conștienți de faptul că dislocînd capul riscați să nu puteți citi corect programele înregistrate în prealabil.

Cunoscînd caracteristicile casetofonului care s-a livrat cu calculatoarele **PRAE,** am prevăzut în continuare o a doua înregistrare pe aceeași față a benzii.

Cea de a doua înregistrare este efectuată cu viteză de transfer mai redusă (calculatorul **PRAE** permite acest lucru). Înainte de a încerca să efectuați citirea înregistrării făcută cu viteză de transfer redusă (1800 bit/sec. spre deosebire de 2500 bit/sec. a înregistrărilor normale) va trebui să modificați parametrul de viteză (adresa **&4018**) al interfeței de casetă. Operația se poate efectua fie din **BASIC PRAE** folosind comanda :

```
POKE &4018,&10
```

fie din monitorul rezident al calculatorului, PRAE, folosind comanda de substituire în memorie :

```
!S4018 0A 10
```

După ce veți fi modificat acest parametru, reluați încercările de încărcare descrise mai sus, știind că veți avea de așteptat mai mult, circa **2'42''**, pentru derularea benzii. Dacă nici cu cea de a doua înregistrare nu veți putea scăpa de erori (TAPE ERROR) atunci va trebui să reglați capul de redare, sau să apelați la un alt casetofon.

Înțelegînd supărarea Dvs. (dacă veți fi ajuns în situația de sus) trebuie să vă precizăm faptul că, caseta Dvs. a fost elaborată pe baza unei tehnologii meticulos elaborate și verificate, de către specialiști ai Electrecord, și ea satisface într-adevăr condițiile calitative necesare pentru a putea fi citită fără erori pe un casetofon standard.

Celor care vor fi avut dificultăți la încărcarea programului, ca și celorlalți de altfel, le recomandăm să-și creeze imediat, după o încărcare reușită, o înregistrare de rezervă (pe altă casetă) folosind comanda :

SAVE "VISIBLE—Z80" &7000, &E000

Dacă au existat probleme la citirea casetei originale, pe un casetofon calificat de Dvs. drept bun, atunci ele vor fi probabil eliminate prin folosirea înregistrării create de acel casetofon.

Vă mai amintim că la citirea oricărei înregistrări magnetice pot apare și erori aleatoare, care trebuiesc eliminate prin încercări succesive. (La dischetele magnetice flexibile se efectuează pînă la 32 de iterații înainte de a se da mesajul de eroare. Din păcate, la casetofonul muzical această operație nu poate fi implementată, datorită imposibilității de a da comanda sensului și vitezei de derulare a benzii precum și timpului de redare mult superior).

Iată-ne ajunși în situația unei încărcări corecte. Programul **VISZ80** se lansează în execuție prin comanda :

CALL & 7C11

Pe ecran va apare Im. 218, imaginea de „trezire" a programului **VISZ80** (vezi Im. 218) și se emite un semnal sonor. După aceea se vor lista comenzile disponibile ale monitorului și programul trece în bucla de așteptare a unei comenzi. În acest moment **VISZ80** se poate considera instalat.

Im. 218

A.1.2. Comenzi pentru calculatorul aMIC

Recomandăm lansarea comenzii de încărcare din monitorul rezident al calculatorului. Caseta fiind poziționată în amonte de înregistrarea de pe fața 2, tastați comanda :

L (CR)

și apoi porniți casetofonul pe poziția REDARE(PLAY).

Dacă după încărcare, revenirea în programul monitor se face fără mesaje de eroare, **VISZ80** se poate lansa în execuție cu comanda : G8017 **(CR)**.

Dacă apar erori la încărcare, vă recomandăm să efectuați verificările și încercările descrise mai sus la prezentarea comenzilor pentru calculatorul **PRAE**.

Pe fața doi s-a generat o singură înregistrare, datorită faptului că **aMIC** nu permite înregistrarea și citirea programelor cu viteză programabilă. Crearea a două înregistrări de tip aMIC ar fi depășit spațiul util al casetei.

Și în acest caz generarea unei înregistrări de rezervă este recomandabilă, imediat după prima citire reușită (salvînd pe o altă casetă conținutul spațiului de memorie cuprins între adresele 7000_H și $E000_H$), fiindcă orice casetofon poate înghiți accidental banda. Pentru acest din urmă caz comentariile sînt de prisos.

A.2. Comenzile monitorului încorporat

Lista comenzilor oferite de monitorul încorporat-ordonate alfabetic- apare în imaginea de trezire a programului VISZ80 (vezi Im. 219).

Fiecare comandă constă dintr-o literă urmată opțional de unul sau mai mulți parametri. Toate numerele care apar ca parametri ai unei comenzi sînt numere hexazecimale. Tastarea unor caractere ilegale va duce la abortarea comenzii (înainte de lansare), eveniment pe care monitorul îl va semnala afișînd "*".

A.2.1. Comenzi generale

Comanda H (help)

Tastînd litera H se afișează meniul de comenzi al monitorului încorporat.

Comanda D (Dump memory)

Comanda D permite vizualizarea (sub forma unor numere hexazecimale) a conținutului memoriei cuprinsă între două adrese limită. Se afișează pînă la opt octeți pe linie. La începutul liniei se afișează adresa de memorie a primei locații de pe rîndul respectiv. Formatul comenzii : **Dadr1, adr2 (CR)**.

— **adr1** este adresa de început, un număr hexazecimal cu 4 cifre semnificative
— **adr2** este adresa de sfîrșit, un număr hexazecimal cu 4 cifre semnificative
— parametri pot fi separați prin virgulă sau spațiu
— între **D** și **adr1** nu se va pune spațiu
— comanda se termină tastînd retur de car (CR)
— dacă s-a comis o eroare la introducerea unuia din parametri, nu este necesară corectarea ei, căci se vor considera totdeauna ultimele 4 cifre hexazecimale.

Un exemplu de utilizare al acestei comenzi se găsește în Im. 224.

Pentru urmărirea mai ușoară a listărilor lungi, după fiecare pagină de ecran afișată, pe rîndul de jos apare în invers video întrebarea :

"TYPE (CR) FOR NEXT PAGE?"

Dacă se tastează retur de car listarea continuă. Pentru orice altă tastă (mai puțin SHIFT sau CTRL) listarea se abortează revenindu-se în secvența de așteptare a unei comenzi) („>").

Comanda F (Fill memory)

Cu această comandă se poate umple o zonă de memorie dată cu o valoare constantă. Formatul comenzii este :

Fadr1,adr2,nn (CR)

— **adr1** este adresa de început a zonei tratate
— **adr2** este adresa de sfîrșit a zonei tratate
— **nn** este un octet exprimat prin valoarea sa hexazecimală
— ca separator de parametri se poate folosi caracterul spațiu sau virgulă

Comanda I (Insert)

Comanda permite înscrierea în memorie la adrese crescătoare a unui șir de valori, începînd cu o adresă dată.

Formatul comenzii este :

Iadr,num

— **adr** este adresa de început a zonei de memorie în care se face inserarea
— **num** este valoarea care se dorește a fi înscrisă în memorie (1 octet)
— pe un rînd se pot scrie mai multe date separate prin virgulă sau punct
— dacă se tastează retur de car (CR) atunci se sare la rîndul următor la începutul căruia se afișează adresa de memorie la care se va face următoarea inserare
— comanda se termină tastînd **ESC** (CTRL—U)
— dacă după fiecare valoare numerică introdusă se tastează CR atunci utilizatorul va vedea fiecare număr în dreptul adresei de memorie, la care acesta se inserează.

Comanda S (Substitute)

Comanda **S** permite actualizarea unei zone de memorie, începînd cu o adresă specificată în lista comenzii. Ea va afișa conținutul curent al celulelor, fiecare valoare putînd fi substituită (opțional) cu alta.

Formatul comenzii :

Sadr,

adr este adresa de început a zonei RAM care urmează să fie tratată.

După adr se tastează „spațiu" sau „virgulă". În acest moment monitorul afișează valoarea curentă memorată în locația respectivă urmată de o liniuță. Dacă se dorește substituirea valorii respective se tastează noua valoare urmată de separator ("," sau " "). După tastarea separatorului se afișează conținutul celulei următoare. Dacă nu se dorește modificarea unei celule, se tastează separatorul.

Comanda se termină cu tasta **CR.**

Comanda X (eXamine or modify registers)

Comanda **X** se poate lansa în două moduri :

X(CR) și
Xreg,num

Prima formă se folosește pentru vizualizarea conținutului tuturor regiștrilor interni ai microprocesorului, sub formă hexazecimală. Se listează conținutul regiștrilor : **B, C, D, E, H, L, A, F, B', C',D', E', H' , L', A', F', I, R, IX, IY, SP, PC.**

A doua formă se folosește pentru vizualizarea și (opțional) modificarea conținutului unui registru selectat.

— **reg** este numele registrului format obligatoriu din 2 caractere : 2 litere, literă și apostrof, literă și spațiu (pentru cele cu nume format dintr-un singur caracter) ; După tastarea numelui, monitorul afișează valoarea curentă a regiștrului urmată de o liniuță "—".

— **num** este valoarea (opțional) tastată pentru a modifica conținutul regiștrului selectat ; el are valoarea exprimată obligatoriu pe 2 sau 4 cifre hexazecimale (2 la regiștri simpli și 4 la regiștri dubli : **PC, SP, IX și IY**) ;

Tastările eronate nu pot fi corectate pe parcursul introducerii. În acest scop se va relansa comanda. (Această rigiditate derivă din formatul în care se memorează conținutul regiștrilor interni ai microprocesorului, în **VISZ80**).

Comanda se termină fie automat, după cele 2 (respectiv 4) cifre tastate, fie tastînd **CR**. (În cazul în care nu se dorește modificarea conținutului curent al registrului selectat).

A.2.2. Comenzi de editare

Comanda A (Assemble)

Comanda **A** invocă asamblorul încorporat în **VISZ80**, care permite introducerea de la tastatură a unor programe scrise în limbajul de asamblare al microprocesorului **Z80** folosind argumente hexazecimale absolute.

Asamblorul se lansează cu comanda :

Aadr (CR)

— **adr** este adresa de memorie începînd cu care se vor depune pe rînd codurile instrucțiunilor tastate.

Asamblorul se anunță prin afișarea adresei de început și preia apoi cîte o instrucțiune pe linie. După introducerea unei linii corecte (terminată prin **CR**) asamblorul depune codul obiect rezultat la adresa dată și revine în modul de editare după afișarea adresei de asamblare a următoarei instrucțiuni.

Ieșirea din asamblor se face prin introducerea unei linii goale (se tastează numai **CR**).

— Asamblorul din **VISZ80** realizează doar traducerea mnemonicelor și numelor regiștrilor în cod binar executabil. Nu permite utilizarea etichetelor, a simbolurilor și cea a pseudoinstrucțiunilor.

— La introducerea unei linii eronate **VISZ80** afișează pe rîndul următor semnul " ?" și cere repetarea liniei curente.

— În cursul editării unei linii, se pot folosi următoarele caractere de control :

TAB(CTRL—I) — introduce spații pînă la o poziție multiplu de 8.

BS(CTRL—H) și **DEL** — șterg ultimul caracter introdus. Ștergerea se face atît în buffer cît și pe ecran.

— Asamblarea instrucțiunilor de salt relativ **JR, JR cc, DJNZ** se face prin specificarea adresei absolute a instrucțiunii la care se va efectua saltul.

Asamblorul va calcula deplasamentul și-l va depune în memorie în cîmpul instr ucțiunii respective.

Considerînd forma liniei sursă

(adr) **(mnem)** **(adrsalt)** (CR)

se va respecta relația

(adr + 2) − 128 < adr.salt < (adr + 2) + 127

unde **adr+2** este adresa instrucțiunii imediat următoare.

În caz de neîncadrare, asamblorul va semnala eroarea, afișînd " ?" și invitînd astfel utilizatorul să introducă o linie corectă.

Exemplu :

```
A7000 (CR)
7000      JR      7005
7002      JR      7085
?
7002      JR      7000
7004
```

— Semnalăm o mică anomalie a asamblorului încorporat în **VISZ80** : constantele care încep cu cifre hexazecimale literale (A—F) trebuie precedate de 0. În caz contrar asamblorul nu semnalează eroare, dar va genera cod eronat.

Exemplu :

corect	**incorect**
LD A,0F2	**LD A,F2**

în memorie se va încărca **02** în loc de **F2**.

Comanda L

Comanda **L** invocă dezasamblorul, încorporat în **VISZ80** care va dezasambla codul binar executabil, locat într-o zonă de memorie specificată și va lista mnemonicele rezultate pe ecran.

Forma comenzii :

Ladr1,adr2 (CR)

— **adr1** și **adr2** sînt adresele de început respectiv sfîrșit zonă de dezasamblat. **L** va lista programul organizat pe linii (o instrucțiune/linie). Structura unei linii listate este :

(adr) (cod obiect) (mnemonică) (argumente)

— **(adr)** este adresa instrucțiunii curente, reprezentată prin 4 cifre hexazecimale

— **(cod obiect)** cuprinde de la 1—4 octeți, care reprezintă codul instrucțiunii listat în valori hexazecimale.

— **(mnemonica)** — mnemonica instrucțiunii dezasamblate, în limbaj Z80

— **(argumente)** — pot fi nume de regiștri sau valori hexazecimale.

Pentru a putea întrerupe listările lungi, după afișarea unei pagini ecran, se va afișa în invers video mesajul :

"TYPE (CR) FOR NEXT PAGE ?"

Dacă se tastează retur de car **(CR)**, listarea continuă. La apăsarea oricărei alte taste (mai puțin SHIFT și CTRL) ea va fi abortată, revenindu-se în bucla principală a monitorului.

Cunoaștem o anomalie (constructivă) a dezasamblorului. Instrucțiunile **EX (SP),HL ; EX (SP),IX ; EX SP,(IY) ; IN A, (n)** și **OUT(n),A** apar la dezasamblare fără paranteze : **EX SP,HL ; EX SP,IX ; EX SP,IY ; IN A,n** și **OUT n,A**.

A.2.3. Comenzi de lansare în execuție

Secvențe de program cod mașină, depozitate în memorie pot fi lansate în execuție prin 3 comenzi : **G, U** și **C**.

Comanda G (Go to address)

Este comanda majoră a pachetului **VISIBLE—Z80**. Comanda invocă simulatorul propriu-zis (care este miezul lui **VISZ80**) și permite lansarea în execuția simulată (sub controlul lui **VISZ80**) a unui program, începînd de la o adresă **(adr)**. Se pot stabili opțional pînă la 2 adrese de oprire **breakp1** și **breakp2**.

Formatul comenzii este

Gadr,(breakp1),(breakp2) (CR)

După lansarea comenzii, pe ecran se va desena structura internă a microprocesorului **Z80** și se va trece la execuția simulată a programului lansat într-unul din cele 7 moduri de simulare proprii simulatorului **VISZ80.**
Definiția modurilor de simulare și programarea lor va fi prezentată la descrierea comenzilor dedicate pentru supervizorul încorporat al **VISZ80.**

Adresele de oprire **(breakp)** nu trebuie să fie neapărat adrese de început instrucțiune. Ele trebuiesc înțelese ca puncte de oprire pe adrese : dacă în orice moment al execuției simulate conținutul magistralei de adrese va deveni egal cu valoarea unui breakpoint, atunci execuția va fi abandonată : se actualizează conținutul regiștrilor și se revine în monitor. Astfel execuția unui program poate fi stopată pe orice ciclu mașină extern procesorului. (Exemplu ciclul READ, WRITE, IN, OUT).

Comanda U (Untraced go to address)

Comanda **U** realizează transferul comenzii din **VISZ80** la programul care începe la adresa specificată, îl execută în timp real și trece controlul din nou lui **VISZ80,** la întîlnirea unei adrese de breakpoint.

Formatul comenzii :

Uadr,(breakp1),(breakp2) (CR)

În cazul utilizării acestor comenzi se recomandă ca programul să se termine cu instrucțiunea **JP 8000.** Astfel controlul va fi transferat înapoi la **VISZ80** și fără a se folosi puncte de oprire.

Restricție :

Revenirile din punctele de oprire s-au implementat prin salturi necondiționate (pe 3 octeți). Din această cauză, în unele situații programul poate intra în „bălării"

Exemplu :

```
7000    JP      7100
7003    LD      A,B
70FF    XOR     A       JP RESTART
7100    LD      B,40
7102    LD      IX,4500
```

Dacă punem un breakpoint la adresa **70FF**, acesta va ocupa 3 octeți (70FF—7101). Dacă în acest caz se execută ramura **JP 7100**, atunci codul instrucțiunii **LD B,40** va fi alterat de către adresa de restart a punctului de oprire.

Avantajul oferit de comanda **U** este acela de a permite utilizatorului rularea unor secvențe lungi și deja testate, în timp real. La returnarea comenzii în monitor, este posibilă execuția unei secvențe interesante în regim simulat (comanda **G**).

Comanda C (Call subroutine)

Comanda **C** are același rol ca și comanda **U**. Ea transferă controlul la subrutina aflată în memorie la adresa adr.

Forma comenzii :

Cadr (CR)

La întîlnirea primei instrucțiuni **RET** sau **RET cc** cu condiția adevărată, controlul revine în monitorul propriu-zis **VISZ80.**

Exemplu :

```
A7000 (CR)
7000    LD    B,3
7002    LD    C,41 ;        ; codul lui „A"
7004    CALL  3154          ; console output Prae
7007    DJNZ  7002
7009    RET
700A
C7000
AAA
```

Prezența comenzii **C** ușurează apelul unor subrutine neinteresante, putîndu-se continua apoi execuția simulată.

A.3. Simulatorul grafic propriu-zis

Acest modul constituie esența pachetului de software **VISIBLE—Z80.** El desenează pe ecran structura internă a microprocesorului **Z80** și derulează în timp toate activitățile care au loc în interiorul microprocesorului, atunci cînd el execută oricare din cele 696 instrucțiuni declarate. El se invocă prin comanda G a monitorului.

Nivelul de detaliare a simulării poate varia în limite largi de la un regim mai detailat decît cel tact cu tact, la un regim instrucțiune cu instrucțiune. Gradul de detaliare este programabil. Pentru același grad de detaliare viteza de execuție este de asemeni programabilă.

Activitățile desfășurate pe ecran sînt subliniate prin secvențe sonore : în regimul cel mai detailat (îl numim impropriu tact cu tact — clock cycle — CC) fiecare mișcare ecran este urmată de emiterea unui sunet monoton (BEEP). Începutul ciclurilor mașină mai importante este marcat prin cîte o melodie care are menirea de a sublinia apariția unui eveniment interesant.

Aceste efecte sonore sînt prezente la instalarea pachetului **VISIBLE—Z80** dar există comenzi care permit suspendarea și eventual revalidarea lor. Aceste elemente, care afectează regimul de lucru a simulatorului formează comenzile dedicate supervizorului. În funcție de situația în care se lansează le categorisim în două grupe : comenzi statice și comenzi dinamice.

A.3.1. Comenzile statice ale supervizorului (M)

Tastînd comanda **M** în monitor se afișează meniul de comandă al supervizorului **(Mode menue).** El se poate vedea în Im. 221. Odată intrați în acest meniu putem activa comenzile supervizorului de simulare (se afișează prompterul meniului supervizor "—").

Comanda E (Examine Simulation Switches)

Tastarea acestei comenzi provoacă listarea sintetică a tuturor parametrilor de lucru importanți ai simulatorului.

Modul de simulare (SIMULATION MODE) : M

poate avea șapte valori distincte. Ele determină regimul de lucru al simulatorului :

M=0 — este cel mai detailat mod. Pe ecran se vizualizează toate activitățile interne ale microprocesorului. Rezoluția este mai mare decît cea tact cu tact (îl vom numi impropriu „tact cu tact" — **CC**)

M=1 — blocurile funcționale reprezentate pe ecran se actualizează la sfîrșitul fiecărui ciclu mașină procesor. După fiecare activitate ecran execuția simulată este suspendată pînă la tastarea tastei (CR). De aceea acest regim se numește ciclu cu ciclu — manual **(MC — manual).**

M=2 — se deosebește de **M1** doar prin faptul că procesul de simulare decurge automat (fără tastări CR), lungimea pauzelor dintre pașii succesivi fiind programabilă **(MC—automat).**

M=3 — este de asemenea regim manual. Execuția simulării va fi suspendată la sfîrșitul fiecărui ciclu instrucțiune și va fi reluată la apăsarea tastei (CR). (Regim Instruction Cycle — **IC — Manual).**

M=4 — Impune un regim **IC automat** cu viteză programabilă.

M=5 — Declanșează un regim de simulare numit **trasor (TRACE).** Pe ecran nu se vizualizează decît adresa, mnemonica și codul instrucțiunii care se execută. Viteza de execuție este de asemenea programabilă.

M=6 — Este modul de lucru **rapid (FAST).** Pe ecran nu se vizualizează nimic. Execuția vizibilă putînd fi reluată schimbînd modul de lucru după un **breakpoint** sau folosind una din comenzile dinamice ale supervizorului (o vom prezenta în continuare).

Temporizarea simulării (TIMING) : T

Este un număr exprimat pe un octet care determină lungimea unei pauze dintre 2 activități ecran în oricare din cele 5 regimuri automate (excepții fac cele manuale M=1 și M=3) la care acest parametru este lipsit de sens). Se poate programa orice valoare cuprinsă între 0 — FF_H, dar supervizorul le va rotunji în jos la una din valorile : **30H, 70H, B0H, F0H** pentru a le corela cu posibilitățile oferite de cursorul de viteză **SPEED** (vezi comenzile dinamice).

Deplasamentul locării programului (DISPLACEMENT): D

A fost prevăzut cu o posibilitate de a relativiza adresele de memorie. Exemplu : dacă deplasamentului i se acordă valoarea 5000H atunci un program locat începînd cu această adresă va fi văzut de către simulator la adresa virtuală 0.

Sunetul (BELL): B

Este un comutator software care validează sau inhibă sunetul la fiecare activitate ecran (B=1 activează, B=0 inhibă).

Cîntecele (SING): S

Este comutatorul software care validează sau inhibă generarea melodiilor la începutul ciclurilor mașină semnificative (S=1 validează, S=0 inhibă).

Ciclurile subliniate cu melodii sînt :

— începutul instrucțiunii (ciclul FETCH)
— ciclul refresh (REFSH)
— acceptarea unei întreruperi nemasabile (NMI)
— acceptarea unei întreruperi mascabile (INT)
— începutul unui ciclu (RESET)
— stările suspendate (BUSAK)
— încercarea de a scrie într-o zonă de memorie protejată
— atingerea unei adrese de breakpoint.

Frecvența sunetului și rapiditatea melodiilor sînt proporționale cu viteza de simulare (T). Un exemplu de folosire al acestei comenzi se poate vedea în Im. 221. Revenirea din meniul supervizorului în bucla monitor se face tastînd (CR).

A.3.2. Comenzile dinamice ale supervizorului

Aceste comenzi se activează în timpul execuției simulate a unui program (lansat cu G). Ele se emit apăsînd mai scurt sau mai lung una din următoarele taste :

M — provoacă abandonarea simulării și revenirea în monitor (">").

B — simulează activarea semnalului $\overline{\text{BUSRQ}}$. Dacă se apasă B în modul de lucru M=0, $\overline{\text{BUSRQ}}$ se va afișa în invers video. Microprocesorul va intra în starea suspendată (HOLD) la încheierea ciclului mașină în curs de simulare. Starea HOLD este marcată de o melodioară specifică și afișarea în invers video a semnalului $\overline{\text{BUSAK}}$. Ieșirea din starea HOLD se face reapăsînd tasta B.

R — simulează aplicarea semnalului $\overline{\text{RESET}}$. Se va genera o melodie specifică. Se simulează activitățile intreprinse de microprocesor în cazul apariției

semnalului **RESET**, după care procesul de simulare va continua de la adresa 0000_H.

I — simulează un semnal de întrerupere $\overline{\text{INT}}$. **VISZ80** comunică acceptarea comenzii prin trecerea în invers video a semnalului $\overline{\text{INT}}$. Intreruperea mascabilă va fi acceptată la sfîrșitul instrucțiunii în curs de execuție, doar dacă **IFF1** era **setat** IFF1=1).

N — simulează generarea unei întreruperi nemascabile. **VISZ80** va trece în invers video semnalul $\overline{\text{NMI}}$. La sfîrșitul instrucțiunii curente întreruperea va fi acceptată.

1,9 — sînt cele două taste de comandă, a barei de viteză SPEED. Cu ajutorul lor se pot modifica concomitent parametrii M—mod și T—temporizare.

Bara notată cu **SPEED** asupra căreia se găsește un cursor, indică viteza și modul de simulare ales pentru simulatorul propriu-zis. Pentru fiecare din cele 5 regimuri de simulare principale **(CC, MC, IC, TRACE, FAST)** pe bara de viteză s-au rezervat segmente. Poziționarea cursorului (folosind tastele 1 și 9) va impune selectarea modului de simulare adecvat. Segmentele sînt subdivizate. Aceste subdiviziuni reprezintă o măsură a temporizărilor.

Regimurile manuale **(M=1** și **M=3)** sînt marcate pe bara **SPEED** prin îngroșarea subdiviziunilor aferente. Trecerea de la regim manual la automat și invers, se face implicit, odată cu mișcarea cursorului, eveniment semnalat de supervizor prin mesajele **AUT** sau **MAN** dispuse în dreapta barei **SPEED.**

A.4. Exemple de lucru în ambianța VISIBLE—Z80.

Vom prezenta în continuare cîteva secvențe de lucru cu ajutorul imaginilor generate prin însuși **VISZ80.**

În Im. 219 se remarcă meniul monitor, urmat de prompterul ">" monitorului. Urmează o secvență de utilizare a asamblorului, lansat cu comanda **A7000(CR).** În linia notată cu **700A** s-a tastat CR revenindu-se în monitor.

În Im. 220 se verifică codul asamblat, folosind comanda **L7000,700A(CR).** Urmează exemplificarea comenzii H și a comenzii I. Cu ajutorul acesteia din urmă se fac două inserări (la adresele **7100** și **7200).** În partea de jos a imaginii se exemplifică modul de utilizare a comenzii **X.** Im. 220 se termină cu utilizarea comenzii **D** pentru verificarea inserărilor făcute cu comanda **I.**

În Im. 221 remarcăm lansarea comenzii **M,** urmată de afișarea **meniului supervizor (MODE MENUE).** Se ilustrează utilizarea comenzilor statice supervizor **E** și **S.** În ultima linie se lansează în execuție cu **breakpoint la 7009,** (folosind comanda de acționare a simulatorului, **G),** programul elaborat în Im. 219.

În Im. 222 se imortalizează sfîrșitul primei instrucțiuni din program, iar în Im. 223 regăsim starea ecranului după atingerea adresei de oprire **(7009).** Regiștrii sînt actualizați, conform programului executat și oprirea este semnalată prin mesajul „**STOP AT BREAKPOINT**".

Revenirea în monitor se semnalează prin afișarea prompterului ">".

În fine, ultima imagine, Im. 224, exemplifică o serie de comenzi monitor în care le remarcăm pe cele notate cu **F, S** și **X.**

```
Z80 VISIBLE SIMULATOR

MONITOR MENU

A -ASSEMBLE          I -INSERT
C -CALL SUBROUT.     L -LIST A PROGRAM
D -DISPLAY MEMORY    M -MODE MENU
F -FILL MEMORY       S -SUBSTITUTE MEMORY
G -GOTO ADDRESS      U -UNTRACED GOTO
H -HELP              X -EX. OR MODIFY REG

>A7000
7000  XOR    A
7001  LD     A,(DE)
7002  ADC    A,(HL)
7003  DAA
7004  LD     (HL),A
7005  INC    E
7006  INC    H
7007  DJNZ   7001
7009  RET
700A
>
```

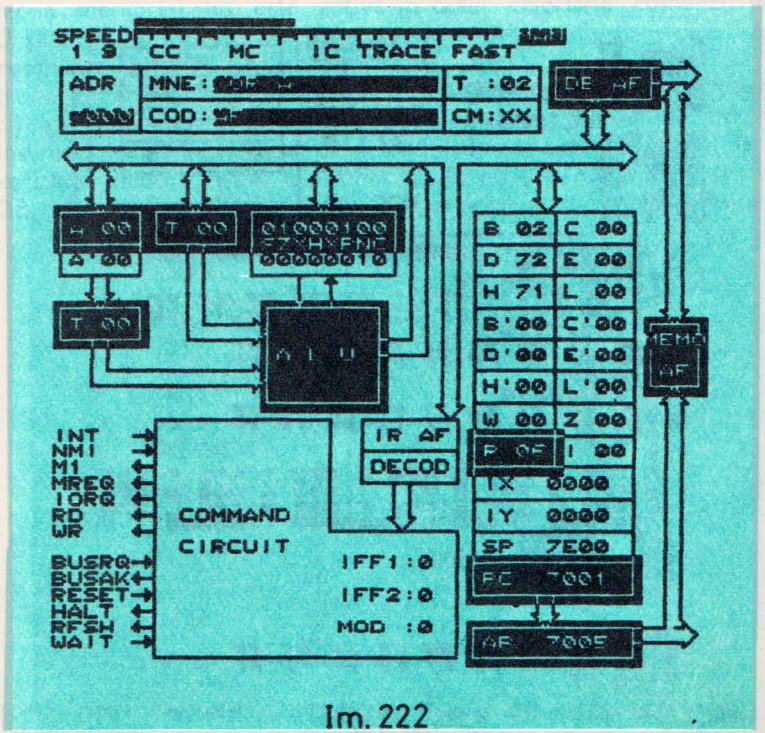
Im.219

```
700A
>L7000,7009
7000  AF      XOR  A
7001  1A      LD   A,(DE)
7002  8E      ADC  A,(HL)
7003  27      DAA
7004  77      LD   (HL),A
7005  1C      INC  E
7006  2C      INC  L
7007  10F8    DJNZ F8
7009  C9      RET
>H
MONITOR MENU

A -ASSEMBLE          I -INSERT
C -CALL SUBROUT.     L -LIST A PROGRAM
D -DISPLAY MEMORY    M -MODE MENU
F -FILL MEMORY       S -SUBSTITUTE MEMORY
G -GOTO ADDRESS      U -UNTRACED GOTO
H -HELP              X -EX. OR MODIFY REG

>I7100
7100 01
7101 99
7102 $
>I7200
7200 99
7201 00
7202 $
>XH  00-71
>XL  00-00
>XD  00-72
>XE  00-00
>XB  00-02
>D7100,7102
7100 01 99 DF
>D7200,7202
7200 99 00 DF
>
```

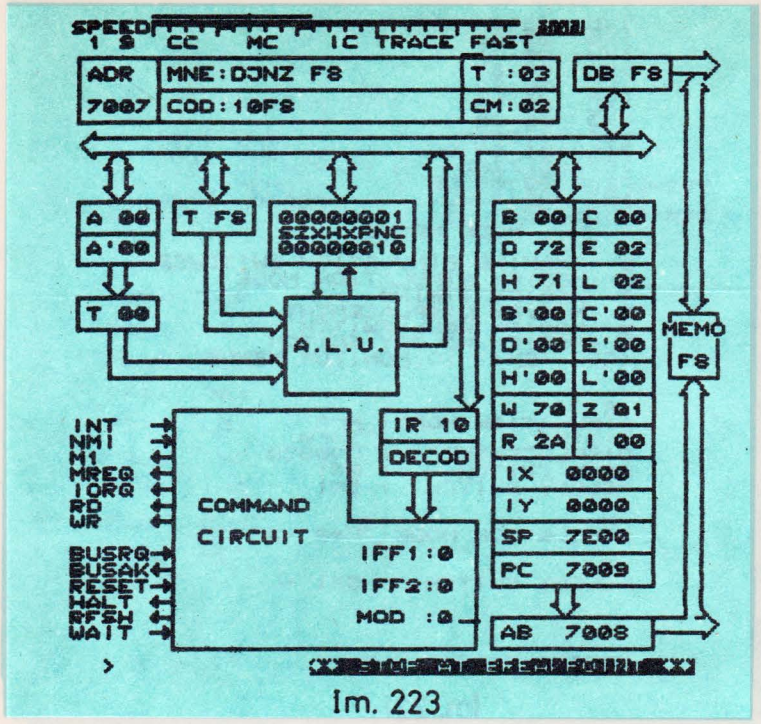
Im.220

```
>XD   72-00
>XB   02-00
>XH   00-71
>XL   00-00
>XD   00-72
>XE   00-00
>XB   00-02
>D7100,7102
7100  01 99  DF
>D7200,7202
7200  99 00  DF
>M

          MODE MENUE

E  -EXAMINE SIMULATION SWITCHES
M  -MODIFY SIMULATION MODE
T  -MODIFY TIMING
D  -MODIFY DISPLACEMENT
B  -MODIFY BELL SWITCH
S  -MODIFY MELODY SWITCH
<CR>  -EXIT TO MONITOR MENUE

-E
SIMULATION MODE:  M=0
TIMING          :  T=F0
DISPLACEMENT    :  D=0000
BELL SWITCH     :  B=1
MELODY SWITCH   :  S=1

-M
SIMULATION MODE:  M=0 -3

-S
MELODY SWITCH   :  S=1 -4

-

>G7000,7009
```

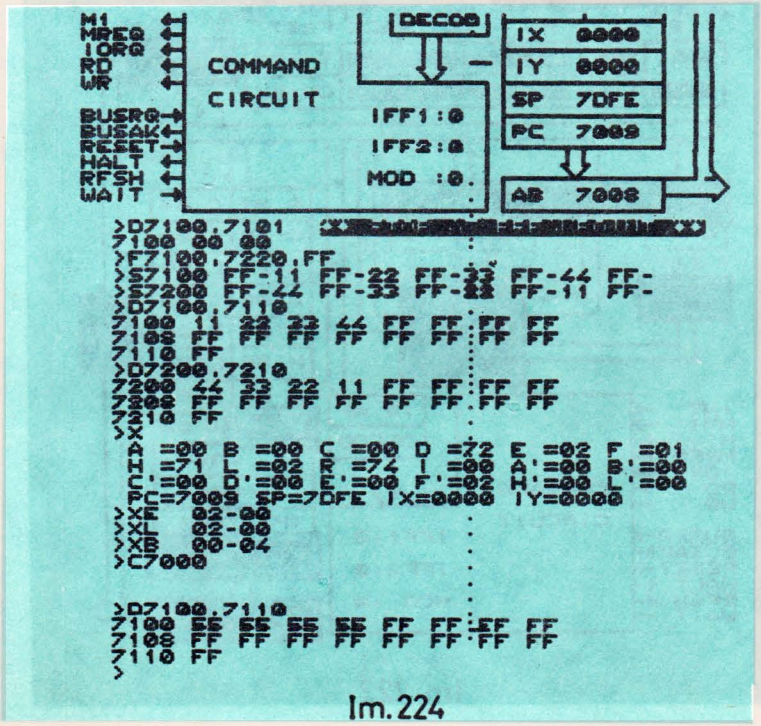
Im. 221



Im. 222


SPEED
1 S CC MC IC TRACE FAST
ADR
7007
MNE:DJNZ FS
COD:10FS
T :03
CM:02
DB FS
A 00
A'00
T FS
00000001
SZXHXPNC
00000010
B 00 C 00
D 72 E 02
H 71 L 02
B'00 C'00
D'00 E'00
H'00 L'00
W 70 Z 01
R 2A I 00
IX 0000
IY 0000
SP 7E00
PC 7009
T 00
A.L.U.
MEMO
FS
IR 10
DECOD
INT
NMI
M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
IFF1:0
IFF2:0
MOD :0
AB 7008
>

Im. 223


M1
MREQ
IORQ
RD
WR
BUSRQ
BUSAK
RESET
HALT
RFSH
WAIT
COMMAND
CIRCUIT
DECOD
IFF1:0
IFF2:0
MOD :0.
IX 0000
IY 0000
SP 7DFE
PC 7009
AB 7008
>D7100,7101
7100 00 00
>F7100,7220,FF
>S7100 FF-11 FF-22 FF-33 FF-44 FF-
>S7200 FF-44 FF-33 FF-22 FF-11 FF-
>D7100,7110
7100 11 22 33 44 FF FF FF FF
7108 FF FF FF FF FF FF FF FF
7110 FF
>D7200,7210
7200 44 33 22 11 FF FF FF FF
7208 FF FF FF FF FF FF FF FF
7210 FF
>X
A =00 B =00 C =00 D =72 E =02 F =01
H =71 L =02 R =74 I =00 A'=00 B'=00
C'=00 D'=00 E'=00 F'=02 H'=00 L'=00
PC=7009 SP=7DFE IX=0000 IY=0000
>XE 02-00
>XL 02-00
>XB 00-04
>C7000
>D7100,7110
7100 55 55 55 55 FF FF FF FF
7108 FF FF FF FF FF FF FF FF
7110 FF
>

Im. 224

A.5. Recomandări pentru utilizarea pachetului VISIBLE—Z80.

Caseta de față poate fi utilizată de către oameni cu pregătire profesională foarte variată : începînd cu cei novici și terminînd cu utilizatorii profesioniști.

Profesioniștii vor putea folosi VISIBLE—Z80 pentru a-și pune la punct programele elaborate cu asambloare mai performante. Ei vor beneficia de comenzile propriu zise ale monitorului, și de cele de lansare **U** și **C**.

Cei care cunosc deja cîte ceva despre microprocesorul **Z80** vor putea să-și asambleze instrucțiunile interesante folosind comanda **A** iar apoi să urmărească execuția lor simulată cu ajutorul comenzii **G**.

Novicilor le recomandăm să lanseze, imediat după instalare, comanda **G7000**, simulatorul, iar după cîteva minute în care totul li se va părea de neconceput, îi invităm să parcurgă treptat, filă cu filă această carte și să recurgă la serviciile casetei ori de cîte ori vor simți nevoia aprofundării celor citite.

În fig. A.1. redăm harta memoriei celor două calculatoare personale **PRAE** și **aMIC**, avînd pachetul **VISIBLE—Z80** instalat. Din figură rezultă faptul că utilizatorul își poate crea programe în următoarele zone RAM.

PRAE : 4400_H — $7F00_H$; aMIC : 6100_H — $7F00_H$ și $E000_H$ —$FFFF_H$

Protecția memoriei.

În timpul simulării unui program cu ajutorul comenzii **G**, zonele **ZPU** sînt singurele zone neprotejate la scriere. La întîlnirea unei operații de scriere la o locație de memorie protejată, simularea se întrerupe și se afișează mesajul :

****R/O MEMORY AREA****

Neprevederea acestei măsuri de protecție ar putea duce la autodistrugerea a însuși simulatorului.

Protecția memoriei nu este activă în cazul celorlalte comenzi monitor : **r, l, A, U** și **C**.

Instrucțiunile de I/O

Pentru a preîntîmpina efecte nedorite (blocarea sistemului, pierderea imaginii, etc.) la simularea accidentală sau de test a instrucțiunilor de intrare/ieșire, acestor instrucțiuni li s-a prevăzut doar modul de execuție pe ecran, fără ca ele să fie executate efectiv. Astfel la instrucțiunile de tip **OUT** data se transferă pînă la **DB**, iar instrucțiunile de tip **IN** de pe orice port vor forța valoarea 55_H.

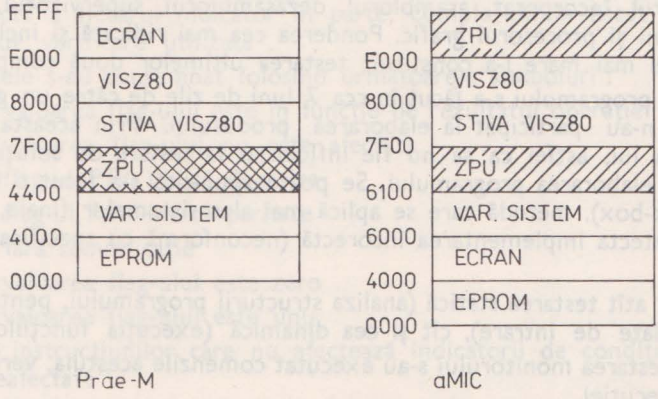


Fig. A.1. Harta memoriei calculatoarelor Prae și aMic, avînd VISZ80 instalat

B

PROCEDURA DE TESTARE PENTRU VALIDAREA PROGRAMULUI DE SIMULARE VISIBLE—Z80

S-a insistat mult asupra testării programului Visible—Z80, avînd în vedere importanța punerii la dispoziția utilizatorului a unui program cît mai corect posibil. Am fost conștienți de faptul că testarea indică prezența erorilor — nu absența lor — și are dezavantajul că pentru un program complex este incomodă, necesitînd multe date și timp excesiv. Totuși, metoda folosită are avantajul că programul se execută în mediul său real, corespunzător utilizării lui finale. Pe această cale este totodată posibilă evaluarea performanțelor programului tratat.

B.1. Obiectivul propus și metoda folosită

S-a urmărit testarea tuturor componentelor software ale mediului **Visible—Z80**: monitorul încorporat, asamblorul, dezasamblorul, supervizorul, simulatorul de instrucțiuni și procesorul grafic. Ponderea cea mai ridicată și inclusiv volumul de muncă cel mai mare l-a constituit testarea ultimelor două componente.

Testarea programului s-a făcut în cca 7 luni de zile de către un grup de specialiști care n-au participat la elaborarea produsului. Prin aceasta s-a asigurat independența lui, astfel ca să nu fie influențat la testare de soluțiile adoptate și aplicate la elaborarea programului. Se poate spune că s-a făcut o testare funcțională (black-box), metodă care se aplică mai ales sistemelor finale, avînd avantajul de a detecta implementarea incorectă (neconformă cu specificațiile) a unor funcții.

S-a făcut atît testarea statică (analiza structurii programului, pentru selectarea setului de date de intrare), cît și cea dinamică (execuția funcțiilor dedicate).

Pentru testarea monitorului s-au executat comenzile acestuia, verificînd corectitudinea execuției.

Asamblorul s-a testat cu ocazia introducerii fiecărei instrucțiuni în vederea testării simulatorului.

Dezasamblorul s-a verificat pe parcursul testării simulatorului, acesta afișînd în colțul din stînga sus a imaginii mnemonica instrucțiunii dezasamblate.

Supervizorul s-a testat modificînd regimurile de lucru pentru una și aceeași instrucțiune executată și comparînd rezultatele între ele precum și cu cele teoretice. În cadrul testării supervizorului s-a controlat și buna funcționare a procedurilor de simulare hardware implementate (BUSRQ, INT, NMI, etc.).

Pentru verificarea simulatorului de instrucțiuni și a procesorului grafic s-a conceput un cadru unitar de testare. Astfel s-a elaborat o fișă de test tip. Fișa s-a completat la testarea fiecărei instrucțiuni a microprocesorului Z80. Instrucțiunile s-au testat în ordine alfabetică.

Iată conținutul fișei de test tip :

1. Instrucțiunea MNEM COD MCy SCy SIMCy

În această rubrică s-au trecut atît valorile teoretice preluate din documentațiile de referință (9) și (25) precum și cele determinate cu ocazia rulării instrucțiunii cu ajutorul lui Visible—Z80.

Semnificația simbolurilor folosite este :

MNEM — mnemonica instrucțiunii
COD — codul instrucțiunii
MCy — numărul de cicluri mașină (machine cycle)
CCy — numărul de cicluri tact (clock cycle)
SIMCy — numărul de secvențe grafice proprii lui Visible—Z80.

2. Conținutul inițial al elementelor implicate

Pentru a urmări buna funcționare a simulatorului de instrucțiuni, s-au ales valori numerice adecvate pentru regiștri implicați în instrucțiunea testată.

3. Conținutul final al elementelor implicate

Asemănător punctului anterior, s-au completat valorile finale (datele de ieșir e, rezultatele) ale tuturor elementelor afectate de instrucțiunea testată ;

4. Flaguri afectate : S Z X H X P N C

S-au consemnat atît valorile „teoretice", conforme cu specificația microprocesorului, cît și cele consemnate cu ocazia rulării instrucțiunilor.

În cazul instrucțiunilor care afectează indicatorii de condiții (flaguri) s-au efectuat mai multe rulări, cu date de intrare diferite, în vederea verificării poziționării corecte a fiecărui indicator în parte, completînd în acest scop rubrica „procedura de verificare utilizată".

Rezultatele s-au consemnat folosind următoarele simboluri :

! — valoarea flag-ului este în funcție de rezultatul operației
. — valoarea flag-ului nu este afectată
V — flag-ul P/V indică depășire
P — flag-ul P/V indică paritate
X — fără semnificație
0 — valoarea flag-ului este zero
1 — valoarea flag-ului este unu

În cazul instrucțiunilor care nu afectează indicatorii de condiție s-a trecut cuvîntul „Neafectate".

5. Anomalii constatate, descrierea clară a contextului.

Au fost descrise anomaliile constatate, cu specificarea clară a contextului.

6. Data, semnătura

S-a marcat data testării instrucțiunii și a completării fișei de test, precum și semnătura „controlorului de calitate". Rubrică pe cît de banală pe atît de importantă.

Corespondența biunivocă între fișele de test și instrucțiunea testată, precum și o referire simplă a instrucțiunilor verificate, au fost asigurate prin numerotarea fișelor de test.

Simularea uneia și aceleiași instrucțiuni s-a executat consecutiv, de mai multe ori, folosind diferite moduri de simulare, atît pentru verificarea acestora din urmă, cît și datorită faptului că a fost imposibil a se marca toate datele fișei de test, printr-o singură trecere.

Prima trecere s-a făcut obligatoriu în modul de simulare cel mai detaliat (CC). Următoarele rulări s-au efectuat în regimul MC (ciclu mașină) și IC (ciclu instrucțiune).

Dintre referințele bibliografice comparative, utilizate, amintim lucrările lui Rodnay Zacks (9) și dr. Makara Ernőné (25). Prima referință bibliografică s-a dovedit a fi utilă datorită existenței prezentărilor grafice sugestive a execuției instrucțiunilor și a poziționării indicatorilor de condiție. Din cea de a doua lucrare s-a preluat numărul de tacți procesor repartizat pe cicluri mașină. Aceste elemente nu s-au regăsit în prima lucrare.

B.2. Rezultatele testării

Erorile detectate în timpul verificărilor pot fi grupate astfel :

1. Erori generale, care afectează mai multe instrucțiuni :

— poziționarea indicatorilor de condiții se face „mut", la fiecare instrucțiune, fără semnalul sonor specific fiecărei secvențe grafice ;

2. Numărul de tacți și numărul de cicluri mașină nu corespund cu cele indicate în referințele bibliografice utilizate ;
3. Pe ecran apar concomitent mai mulți pași de simulare ;
4. Poziționarea incorectă a indicatorilor de condiție ;
5. Codul instrucțiunii nu se generează corect ;
6. Numerotarea greșită a unui ciclu de tact : de exemplu după tactul T=02, urmează imediat un tact cu numărul T=04.
7. Greșeli ale asamblorului ;
8. Antetul imaginii vizualizate este eronat.

Cîteva erori semnalate la testare s-au dovedit a fi numai restricții de implementare. Explicația acestor aparente anomalii se va da în paragraful următor.

Din totalul de 728 (697+31) instrucțiuni testate, 243 au avut erori care au necesitat o depanare ulterioară.

S-au urmărit aproximativ 19.000 de secvențe grafice, testarea efectuîndu-se în aproximativ 230 de sesiuni de lucru a cîte șase ore.

B.3. Răspunsul elaboratorilor

Observațiile consemnate pe fișele de test au fost analizate de către elaboratorii programului.

Erorile s-au eliminat consemnînd pe spatele fișei respective data și semnătura celui care a făcut corecția.

Unele anomalii semnalate le-am considerat a fi fenomene normale, caz în care am explicat cauza abaterii de la documentația de referință. Iată pe scurt precizările făcute :

a) Statutul de element afectat :

Considerăm un element ca fiind afectat, în timpul unui ciclu sau al unei instrucțiuni, dacă în acest timp conținutul acestui element se schimbă, sau este testat de circuitul de comandă pentru luarea unei decizii. Astfel **F** este un registru afectat de instrucțiunile de salt condiționat. Astfel pe parcursul execuției unei instrucțiuni de tipul **JP cc, nn** conținutul registrului **F** este testat pentru a vedea dacă condiția **cc** este adevărată sau nu, chiar dacă conținutul lui **F** nu se schimbă.

b) La unele cicluri mașină există cicluri de tact pentru care **Visible—Z80** nu arată nimic. Aceste cicluri „vide" apar în două cazuri :

— tacți în care se execută operații interne (exemplu luarea unei decizii de către circuitul de comandă ; tacți de sincronizare). Acești tacți apar de obicei la sfîrșitul ciclurilor mașină. Credem că pe durata acestor tacți în **Visible—Z80** într-adevăr nu trebuie să se întîmple nimic ;

— la incrementarea/decrementarea regiștrilor dubli :

regiștrii dubli precum și registrul **R** se incrementează/decrementează nu prin **ALU** ci prin intermediul unui registru de incrementare/decrementare care nu este reprezentat în modelul grafic al microprocesorului adoptat în **Visible—Z80.** În realitate o incrementare sau decrementare se realizează pe doi tacți : în primul tact conținutul registrului dublu este introdus în registrul de incrementare/decrementare, iar pe al doilea tact conținutul incrementat sau decrementat este transferat înapoi în registrul dublu. Există o regulă : transferul **Reg→reg incr/decr** se realizează pe tacții impari **(T1, T3, T5)** iar transferul înapoi pe tacții pari **T2, T4, T6.** Nereprezentarea în registrul de incrementare/decrementare în modelul grafic adoptat în **Visible—Z80** a dus la apariția unor inexactități în simularea activității de incrementare/decrementare a regiștrilor dubli. Astfel :

— regiștrii dubli se incrementează pe un tact iar celălalt tact rămîne gol

— nu s-a respectat o regulă clară privind momentul în care se simulează incrementarea sau decrementarea. Astfel în ciclul de REFRESH, schimbarea lui **PC** și **R** au loc în **T2** respectiv **T4,** ce corespunde transferului conținutului acestora din regiștrii de incrementare/decrementare înapoi ; dar la **INC rr** schimbarea lui **rr** se simulează pe **T5** ceea ce corespunde momentului de transfer în registrul de incrementare/decrementare.

c) În cazul instrucțiunilor care se descriu pe 3 octeți de cod și un octet de deplasament, ciclul mașină intern — necesare — calculului adresei de indexare se suprapune peste ciclul de citire (fetch) al celui de-al treilea octet cod de instrucțiune. Programarea concurentă a acestor două activități ar fi cauzat probabil ambiguități de interpretare. De aceea **Visible—Z80** tratează cele două activități în cicluri mașină distincte, motiv pentru care numărul de cicluri mașină al acestor instrucțiuni diferă de cel real.

Exemplu :

BIT 7, (IX+IND)
M1 — 4 tacți procesor — fetch DD_H
M2 — 4 tacți procesor — fetch CB_H
M3 — 3 tacți procesor — read IND
M4 — 5 tacți procesor — efectuarea sumei IX+IND
M5 — 4 tacți procesor — citirea octetului specificat, din memorie și execuția instrucțiunii.

Comparînd această realitate cu fenomenologia din Visible—Z80 avem :
Visible—Z80 — 4+4+3+5+4+4 tacți procesor
Real — 4+4+3+5 +4 tacți procesor

d) Nu știm cum se realizează intern instrucțiunile **RLD, RRD.** De aceea un ciclu mașină al acestor instrucțiuni este foarte sărac în **Visible—Z80.**

B.4. Probleme nerezolvate

În ciuda eforturilor depuse au rămas cîteva anomalii, care n-au putut fi eliminate din **Visible—Z80,** datorită în primul rînd lipsei de timp. În cazul unor erori am decis menținerea lor, datorită faptului că le-am considerat mai puțin semnificative, iar corectarea lor ar fi necesitat o restructurare prea amplă a programului.

Anomaliile persistente le-am semnalat cu ocazia aventurilor din partea I a cărții, dar le vom grupa și în cele ce urmează :

a) În cazul instrucțiunilor **SBC HL, rr** unde **rr** poate fi BC, DE, HL sau SP cel de-al treilea ciclu mașină durează patru tacți procesor. În Visible—Z80 durata lui este de 3 tacți procesor.

b) În ciclurile mașină de tip OUT, comun tuturor instrucțiunilor de ieșire **(OUT (n),A; OUT (C),r; OUTD, OTDR, OUTI, OTIR)** semnalul $\overline{WR}$ se activează înainte ca datele de transferat să fie plasate pe magistrala de date. În realitate secvența este inversă (vezi capitolul D. „Datele de catalog ale microprocesorului **Z80"**, diagramele de timp a ciclului **OUT**).

c) La acceptarea unei întreruperi mascabile salvarea valorii curente a contorului program **PC** se face la adresă eronată. În Visible Z80 s-a omis decrementarea prealabilă a lui **SP** (Stack Pointer) ;

d) La intrarea în stare oprită (execuția instrucțiunii **HALT**), Visible—Z80 nu salvează valoarea curentă a contorului program **PC** în registrul **WZ.** Valoarea contorului program **PC** avansează la fiecare pseudo-fetch (**NOP**-uri), iar vechea valoare nefiind memorată nicăieri, execuția programului nu se va putea relua de la instrucțiunea următoare celei **HALT** (la acceptarea unei întreruperi).

e) Dezasamblorul omite parantezele de adresare indirectă la șase instrucțiuni :

	EX	**(SP),HL**		**EX SP,HL**
	EX	**(SP),IX**		**EX SP,IX**
corect	**EX**	**(SP),IY**	incorect	**EX SP,IY**
	IN	**A,(n)**		**IN A,n**
	OUT	**(n),A**		**OUT n,A**

FIŞA DE TEST

SIM Z80

1.Instrucţiunea:

MNEM	COD	MCy	CCy	SIMCy

2. Conţinutul iniţial al elemenetelor implicate:

3. Conţinutul final al elemenetelor implicate:

4. Flaguri afectate: S Z X H X P N C

- Procedura de verificare:utilizată:

5. Anomalii constatate, descrierea clară a contextului:

6. Data, semnătura:

f) În fine amintim o eroare care s-a corectat în program dar ea persistă în imaginile unei aventuri din carte (Im.176—Im.184). Semnalul **IORQ** nu se activează în ciclul de aşteptare a unei întreruperi nemascabile. Rugăm cititorii să ne şcuze pentru imposibilitatea de a mai corecta aceste imagini.

Sîntem conştienţi de faptul că testarea nu garantează corectitudinea absolută a programului, pentru că nu există încă nici o metodă de testare care să demonstreze că programul este fără erori. Testarea ne asigură că programul funcţionează corect pentru setul de date de intrare utilizat. Practica arată însă că testarea statică asociată cu cea dinamică conduce la un program cu un grad ridicat de funcţionalitate.

Faptul, că s-a făcut testarea masivă a programului **Visible—Z80** în sensul că simularea fiecărei instrucţiuni a microprocesorului **Z80** a fost testată chiar de mai multe ori — bineînţeles nu şi toate combinaţiile de argumente posibile, — credem şi sperăm că îi conferă produsului un certificat de calitate şi garanţie. Totuşi în încheiere facem invitaţia către toţi cititorii acestei cărţi şi către toţi utilizatorii programului **Visible—Z80** de a semnala pe o fişă de test eventualele erori descoperite. Vă recomandăm să vă ghidaţi după fişa model fotografiată în fig. B.1., care este o fişă reală, conţinînd atît formularul tip, cît şi completările făcute de cel care a testat instrucţiunea în cauză—fişa nr. 250, pentru instrucţiunea **LD A,R.** Editura şi autorul dispun de toate cele 728 de fişe care au rezultat din testare). Mulţumim.

C

COMPARAȚIA SEMANTICĂ A LIMBAJELOR DE ASAMBLARE I8080 ȘI Z80

Am amintit deja faptul că microprocesorul Z80 „este fratele mai mare" a microprocesorului I8080. El recunoaște și execută întocmai totalitatea instrucțiunilor „fratelui mai mic", posedînd totodată facilitatea de a executa suplimentar alte cîteva sute de instrucțiuni.

La nivelul limbajului de asamblare ale celor două microprocesoare apar diferențe, chiar și în cazul instrucțiunilor comune, al căror cod binar executabil este identic. Din paragraful 3.3 știm deja că structura generală a unei instrucțiuni în limbaj de asamblare este identică pentru cele două procesoare în cauză. Diferențele totuși existente sînt de ordin sintactic și mai ales semantic.

În prezentul capitol vom încerca să extragem și să explicăm aceste diferențe.

Începem cu lista explicitată a mnemonicelor celor două microprocesoare considerate.

Tabela C.1. LISTA MNEMONICELOR I 8080

Nr. crt.	Mnemonica	Semnificația	Nr. de instrucțiuni pe care le descrie
1.	2.	3.	4.
1.	ADC	add with-carry — adună cu transport	8
2.	ADD	add immediate — adună imediat	8
3.	ACI	add with carry immediate — adună cu carry imediat	1
4.	ADI	add immediate — adună imediat	1
5.	ANA	AND with accumulator — SI cu acumulatorul	8
6.	ANI	AND immediate — SI imediat	1
7.	CMA	complement accumulator — complementează acumulatorul	1
8.	CMC	Complement carry — complementează transportul	1
9.	CMP	compare — compară	8
10.	CPI	compare immediate — compară imediat	1
11.	CALL	call — apel	1
12.	CNZ	call if non zero — apel dacă diferit de zero	1
13.	CZ	call if zero — apel dacă zero	1
14.	CNC	call if non carry — apel dacă nu e transport	1
15.	CC	call if carry — apel dacă este carry	1

Tabela C1 (continuare)

1.	2.	3.	4.
16.	CPO	call if parity odd — apel dacă paritatea e impară	1
17.	CPE	call if parity even — apel dacă paritatea e pară	1
18.	CP	call if plus — apel dacă pozitiv	1
19.	CM	call.if minus — apel dacă negativ	1
20.	DAA	decimal adjust accumulator — ajustează BCD acumulatorul	1
21.	DAD	double add — adunare dublă	4
22.	DCR	decrement — decrementează	4
23.	DCX	decrement double register — decrementează registrul dublu	4
24.	DI	disable interrupt — inhibă întreruperea	1
25.	EI	enable interrupt — validează întreruperea	1
26.	HLT	halt — opreşte	1
27.	IN	in — intră	1
28.	INR	increment — incrementează	8
29.	INX	increment double register — incrementează registrul dublu	4
30.	JMP	jump — salt	1
31.	JNZ	jump if non zero — salt dacă diferit de zero	1
32.	JZ	jump if zero — salt dacă zero	1
33.	JNC	jump if non carry — salt dacă nu e transport	1
34.	JC	jump if carry — salt dacă e carry	1
35.	JPO	jump if parity odd — salt dacă paritatea e impară	1
36.	JPE	jump if parity even — salt dacă paritatea e pară	1
37.	JP	jump if plus — salt dacă pozitiv	1
38.	JM	jump if minus — salt dacă negativ	1
39.	LDA	load accumulator — încarcă acumulatorul	1
40.	LDAX	load accumulator from double register — încarcă acumulatorul „după" registrul dublu	2
41.	LHLD	load HL direct — încarcă HL direct	1
42.	LXI	load double register immediate — încarcă registrul dublu imediat	4
43.	MOV	move — mută (transferă)	63
44.	MVI	move immediate — mută (transferă) imediat	8
45.	NOP	no operation — nici o operaţie	1
46.	ORA	OR with accumulator — SAU cu acumulatorul	8
47.	ORI	OR immediate — SAU imediat	1
48.	OUT	OUT — ieşi	1
49.	PCHL	load PC with HL — încarcă PC cu HL	1
50.	PUSH	PUSH on stack — salvează pe stivă	4
51.	POP	POP from stack — restaurează de pe stivă	4
52.	RLC	rotate left with carry — roteşte la stînga cu transport	1
53.	RRC	rotate right with carry — roteşte la dreapta cu transport	1
54.	RAL	rotate left arithmetic — roteşte aritmetic la stînga	1
55.	RAR	rotate right arithmetic — roteşte aritmetic la dreapta	1
56.	RET	return — revenire	1
57.	RNZ	return if non zero — revenire dacă diferit de zero	1
58.	RZ	return if zero — revenire dacă zero	1
59.	RNC	return if non carry — revenire dacă nu e transport	1
60.	RC	return if carry — revenire dacă e transport	1
61.	RPO	return if parity odd — revenire dacă paritatea e impară	1
62.	RPE	return if parity even — revenire dacă paritatea e pară	1
63.	RP	return if plus — revenire dacă pozitiv	1
64.	RM	return if minus — revenire dacă negativ	1
65.	RST	restart — relansare	8
66.	STA	store accumulator — stochează acumulatorul	1
67.	STAX	store accumulator from double register — stochează acumulatorul „după" registrul dublu	2
68.	SHLD	store HL direct — stochează HL direct	1
69.	SUB	subtract — scade	8

Tabela C1 (continuare)

1.	2.	3.		4.
70.	SUI	subtract immediate	— scade imediat	1
71.	SBB	subtract with borrow	— scade cu împrumut	8
72.	SBI	subtract with borrow immediate	— scade cu împrumut imediat	1
73.	STC	set carry	— înscrie transportul	1
74.	SPHL	load SP with HL	— încarcă SP cu HL	1
75.	XCHG	exchange double register	— interschimbă registrul dublu	1
76.	XTHL	exchange top of stack with HL	— interschimbă vîrful stivei cu HL	1
77.	XRA	EXCLUSIVE OR with accumulator	— SAU EXCLUSIV cu acumulatorul	8
78.	XRI	EXCLUSIVE OR immediate	— SAU EXCLUSIV imediat	1
			Total instrucțiuni :	244

Tabela C.2. **Lista mnemonicelor Z80**

Nr. crt.	Mnemonica	Semnificația		Numărul instrucțiunilor pe care le descrie
1.	2.	3.		4.
1.	ADC	add with carry	— adună cu transport	15
2.	ADD	add	— adună	23
3.	AND	AND	— SI	11
4.	BIT	test bit	— testează bit	80
5.	CALL	call	— apel	9
6.	CCF	complement carry flag	— complementează indicatorul de transport	1
7.	CP	compare	— compară	11
8.	CPD	compare & decrement	— compară & decrementează	1
9.	CPDR	compare, decrement, repeat	— compară, decrementează, repetă	1
10.	CPI	compare & increment	— compară & incrementează	1
11.	CPIR	compare, increment, repeat	— compară, incrementează, repetă	1
12.	CPL	complement (acumulator)	— complementează (acumulator)	1
13.	DAA	decimal adjust accumulator	— ajustează BCD acumulatorul	1
14.	DEC	decrement	— decrementează	16
15.	DI	disable interrupt	— inhibă întreruperea	1
16.	DJNZ	decrement, jump if non zero	— decrementează, salt dacă diferit de zero	1
17.	EI	enable interrupt	— validează întreruperea	1
18.	EX	exchange	— interschimbă	5
19.	EXX	exchange X (all)	— interschimbă X (toate)	1
20.	HALT	halt	— oprește	1
21.	IM	interrupt mode	— modul de întrerupere	3
22.	IN	in	— intră	8
23.	INC	increment	— incrementează	16
24.	IND	in & decrement	— intră & decrementează	1
25.	INDR	in, decrement, repeat	— intră, decrementează, repetă	1
26.	INI	In & increment	— intră & incrementează	1
27.	INIR	in, increment, repeat	— intră, incrementează, repetă	1
28.	JP	jump	— salt	12
29.	JR	jump relativ	— salt relativ	5
30.	LD	load	— încarcă (transferă)	132

Tabela C.2. (continuare

1.	2.	3.		4.
31.	**LDD**	load & decrement	— transferă & decrementează	1
32.	**LDDR**	load, decrement, repeat	— transferă, decrementează, repetă	1
33.	**LDI**	load, & increment	— transferă & incrementează	1
34.	**LDIR**	load, increment, repeat	— transferă, incrementează, repetă	1
35.	**NEG**	negative (A)	— negativ (A)	1
36.	**NOP**	no operation	— nici o operație	1
37.	**OR**	OR	— SAU	11
38.	**OTDR**	out, decrement, repeat	— ieși, decrementează, repetă	1
39.	**OTIR**	out, increment, repeat	— ieși, incrementează, repetă	1
40.	**OUT**	out	— ieși	8
41.	**OUTD**	out & decrement	— ieși & decrementează	1
42.	**OUTI**	out & increment	— ieși & incrementează	1
43.	**POP**	POP from stack	— restaurează de pe stivă	6
44.	**PUSH**	PUSH on stack	— salvează pe stivă	6
45.	**RES**	reset (bit)	— resetează (șterge) bit	80
46.	**RET**	return	— revenire	9
47.	**RETI**	return from interrupt	— revenire de la întrerupere	1
48.	**RETN**	return from non mascable interrupt	— revenire de la întrerupere nemascabilă	1
49.	**RL**	rotate left	— rotește la stînga	10
50.	**RLA**	rotate left accumulator	— rotește la stînga acumulatorul	1
51.	**RLC**	rotate left with carry	— rotește la stînga cu transport	10
52.	**RLCA**	rotate left with carry A	— rotește la stînga cu transport A	1
53.	**RLD**	rotate left digit	— rotește la stînga cifră	1
54.	**RR**	rotate right	— rotește la dreapta	10
55.	**RRA**	rotate right accumulator	— rotește la dreapta acumulator	1
56.	**RRC**	rotate right with carry	— rotește la dreapta cu transport	10
57.	**RRCA**	rotate right with carry A	— rotește la dreapta cu transport A	1
58.	**RRD**	rotate right digit	— rotește la dreapta cifră	1
59.	**RST**	restart	— relansare	8
60.	**SBC**	subtract with carry	— scădere cu transport (împrumut)	15
61.	**SCF**	set carry flag	— înscrie indicatorul de transport	1
62.	**SET**	set (bit)	— înscrie (bit)	80
63.	**SLA**	shift left arithmetic	— deplasează aritmetic la stînga	10
64.	**SRA**	shift right arithmetic	— deplasează aritmetic la dreapta	10
65.	**SRL**	shift right logic	— deplasează logic la dreapta	10
66.	**SUB**	subtract	— scade	11
67.	**XOR**	EXCLUSIV OR	— SAU EXCLUSIV	11
			Total instrucțiuni :	696

Dacă comparăm cele două tabele putem face cîteva remarci.

1. Limbajul de asamblare a microprocesorului **Z80** exprimă **696** de instrucțiuni prin **67** de mnemonici, pe cînd cel al microprocesorului **I8080** folosește **78** de mnemonici pentru **244** de instrucțiuni. Conținutul semantic al mnemonicilor **Z80** este mai mare (**10,4** instrucțiuni/mnemonică) decît cel al mnemonicilor **I8080** (**3,1** instrucțiuni/mnemonică).

2. Densitatea de repartiție a instrucțiunilor pe mnemonici este neuniformă, ea variind între limite largi.

18080 Instrucțiuni codificate	total mnemonici	Z80 Instrucțiuni codificate	total mnemonici
cîte : 1	57	cîte : 1	35
cîte : 2	2	cîte : 3	1
		cîte : 5	2
cîte : 4	6	cîte : 6	2
cîte : 8	12	cîte : 8	3
		cîte : 9	2
cîte : 63	1	cîte : 10	7
	78		
		cîte : 11	5
		cîte : 12	1
		cîte : 15	2
		cîte : 16	2
		cîte : 23	1
		cîte : 80	3
		cîte : 132	1
			67

Observăm că limbajul Intel folosește **57** de mnemonici distincte pentru simbolizarea a cîte unei singure instrucțiuni, pe cînd limbajul **Zilog** doar **35.** Numărul maxim de instrucțiuni simbolizate printr-o singură mnemonică este la **Intel 63 (MOV)** iar la **Zilog 132 (LD).**

3. Mnemonicile **Intel** sînt strîns corelate cu lumea hardware a microprocesorului. Distingem cîteva reguli :

— Instrucțiunile ce folosesc adresarea imediată se disting de cele ce folosesc adresarea implicită, chiar dacă instrucțiunile aferente au aceeași funcție. Mnemonicile instrucțiunilor cu adresare imediată se termină cu litera I (immediate). Ex. **ANA, ANI, ADD, ADI, CMP, CPI,** etc.

— instrucțiunile care operează cu regiștri dubli conțin în mnemonică litera **X,** distingîndu-se de instrucțiunile ce operează pe regiștri simpli, chiar dacă funcțiile lor sînt similare. ex. **INR, INX, MVI, LXI, DCR, DCX** etc.

— Instrucțiunile de salt condiționat **(jump, call, ret)** încorporează în mnemonică însăși condiția testată. Această discriminare forțată poate duce la dificultăți de identificare a mnemonicelor : cine își poate închipui că **CC** este de fapt tot o instrucțiune **CALL ? !**

4. Mnemonicele Zilog sînt mai unitare, ele urmărind o clasificare funcțională a instrucțiunilor. Detașîndu-se prin generalitatea lor de detaliile de funcționare hardware a microprocesorului, ele apropie mai mult limbajul de asamblare Z80 de limbajele de nivel înalt, invitîndu-l pe programator la gîndire mai abstractă.

Apare o deosebire și în sintaxa de scriere a operanzilor. Limbajul de asamblare **Zilog** prevede închiderea în paranteze a operanzilor care specifică adresarea indirectă.

Exemple : a) **JP (HL)** — **Zilog**
PCHL — **Intel**

Cele două instrucțiuni sînt identice : se va efectua salt la adresa conținută în **HL.** Mnemonica **Zilog** exprimă logica fenomenului, iar cea Intel fenomenul fizic.

b) **LD B,.(HL)** **LD A,(BC)**
MOV B,M **LDAX B**

Limbajul de asamblare **Zilog** este consecvent în marcarea transferului cu adresare indirectă, indiferent care registru dublu conține adresa de memorie. Limbajul **Intel** tolosește fie operandul **M,** cînd adresarea indirectă se face prin **HL,** fie marchează evenimentul în mnemonica instrucțiunii **(LDAX).** Această incon-

secvență își are și ea explicația : fizic microprocesorul identifică în mod diferit registrul dublu **HL** față de celelalte două **(BC și DE)**.

c) **LD A, (IX + IND)**

Această instrucțiune folosește adresarea indexată pentru a localiza celula de memorie al cărei conținut se va încărca în A. Adresarea indexată fiind în esență indirectă, regula privind folosirea parantezelor se aplică în acest caz. 18080 nu posedă instrucțiuni similare.

Am prezentat comparativ limbajele de asamblare a două microprocesoare, pentru a fixa mai bine noțiunile și caracteristicile legate de aceste limbaje. Ele sînt menite să ușureze munca programatorilor, eliberîndu-i de povara memorării unor coduri.

D

DATELE DE CATALOG ALE MICROPROCESORULUI Z80 (Z80A, Z80B)

În prezentul capitol redăm caracteristicile electrice și cele climatice ale microprocesorului prezentat.

Datele de catalog se defalcă în două clase :

— **Valorile limită absolute**

— **Valorile caracteristice**

În prima clasă se includ acei parametri care nu vor trebui depășiți în nici un caz, fabricantul declinîndu-și orice responsabilitate în acest caz.

Între valorile caracteristice le distingem pe cele statice și pe cele dinamice.

Înșirăm prezentele date, publicate pentru produsul **SGS—Z8400 (Z80),** fiindcă ele sînt prezentate foarte metodic.

Menționăm că proiectanții de hardware, vor trebui să confrunte aceste date cu cele concrete publicate de firma de la care provine procesorul pe care urmează să-l folosească.

A. Valori limită absolute

1. Temperatura ambiantă la funcționare	**0°C— 70°C**
2. Temperatura ambiantă la stocare	**—65°C—150°C**
3. Tensiune pe oricare pin al capsulei față de GND	**—0,3V la +7V**
4. Putere disipată	**1,5 W**

Depășirea oricărei valori de mai sus poate duce la defectarea iremediabilă a microprocesorului **Z80.**

B. Valori caracteristice

D.1. Parametri statici

Condiții; $T_A = 0°C - 70°C$; $V_{cc} = +5\,V \pm 5\%$

Simbol	Parametru	Min.	Tip	Max	UM	Condiții de test
V_{ILC}	Nivel 0 al tactului Ø	−0,3		0,45	V	
V_{IHC}	Nivel 1 al tactului Ø	$V_{cc}-0,62$		V_{cc}	V	
V_{IL}	Nivel 0 la oricare intrare	−0,3		0,8	V	
V_{IH}	Nivel 1 la oricare intrare	2,0		V_{cc}	V	
V_{OL}	Nivel 0 la oricare ieşire			0,4	V	$I_{OL} = 1,8$ mA
V_{OH}	Nivel 1 la oricare ieşire	2,4			V	$I_{OH} = -250$ µA
I_{CC}	Curent de alimentare			150, 200	mA	
				200 *	mA	$t_c = 400$ ns
I_{LI}	Curent absorbit la intrare			10	µA	$0 < V_{IN} < V_{cc}$
I_{LOH}	Curent de scurgere în tri-state			10	µA	$V_{OUT} = 2,4 \div V_{cc}$
I_{LOL}	Curent de scurgere în tri-state			−10	µA	$V_{OUT} = 0,4V$
I_{LD}	Curent de scurgere la DATA			+10	µA	$0 < V_{IN} < V_{cc}$
$C_{Ø}$	Capacitate de intrare pe Ø			20	pF	$T_A = 25°C$ Ø = 1 MHz
C_{IN}	Capacitate de intrare			5	pF	
C_{OUT}	Capacitate de ieşire			10	pF	

*** Z80, Z80A, Z80B**

Pinii nemăsurați sînt legați la masă

D.2. Parametri dinamici

Acești parametri se referă la timpii caracteristici ai diverselor semnale CPU, considerate în diversele cicluri mașină externe procesorului.

Pe figurile D.1—D.8 redăm sub forma unor numere încercuite parametrii dinamici interesanți ale căror valori se găsesc în tabela care completează prezenta anexă.

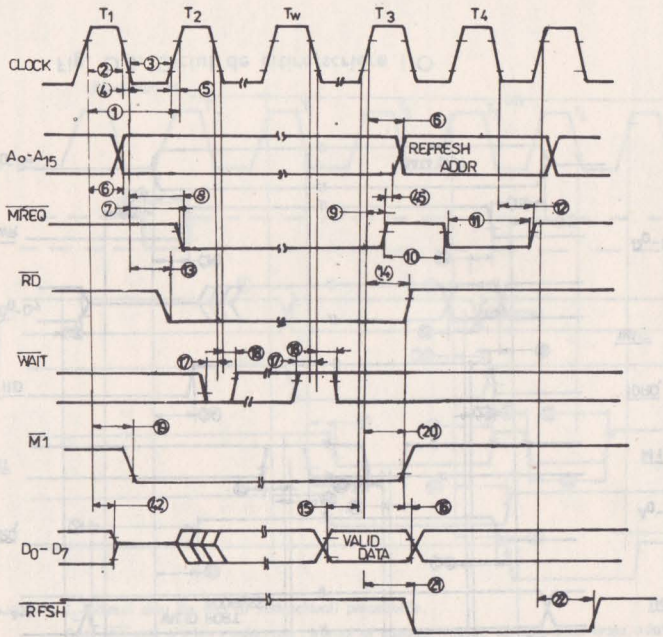


Fig. D.1. Ciclul mașină M1 (fetch)

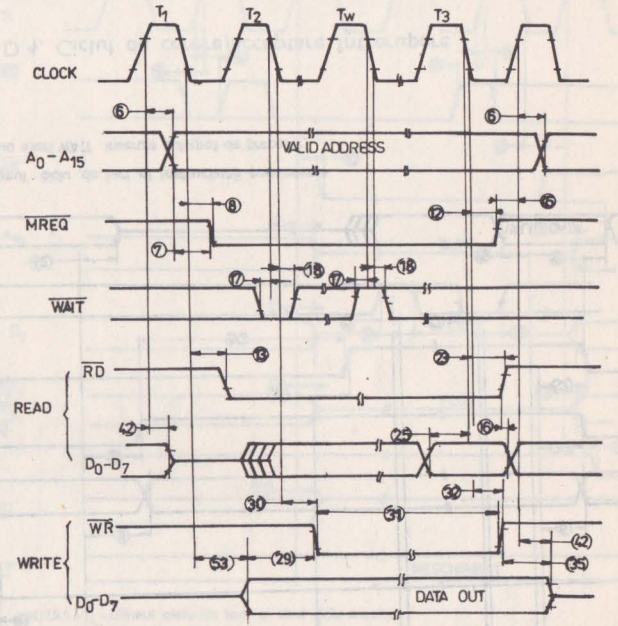


Fig. D.2. Ciclul de citire/scriere memorie

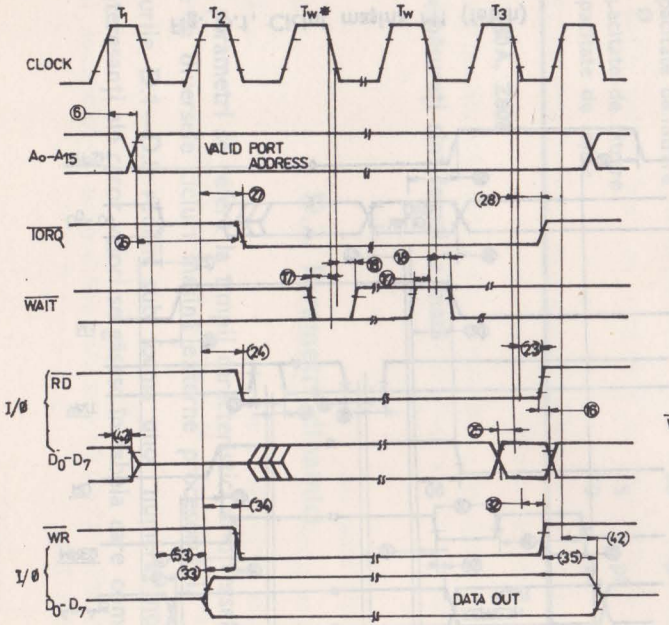
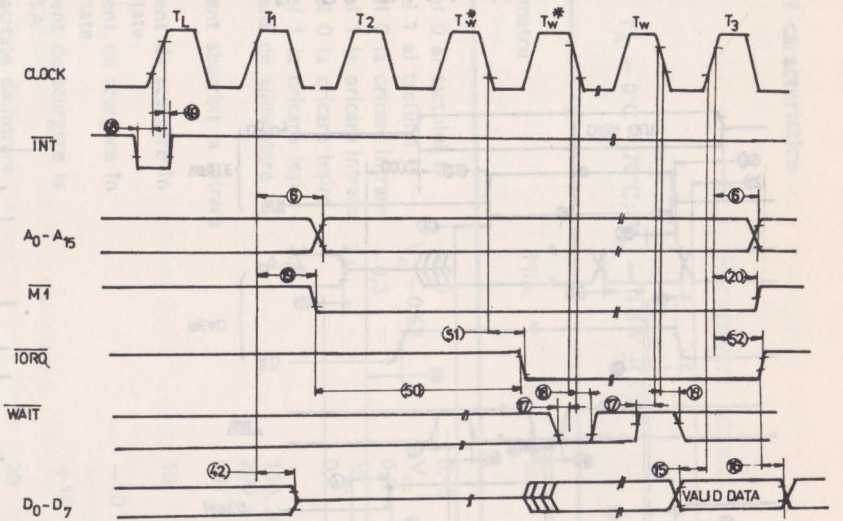
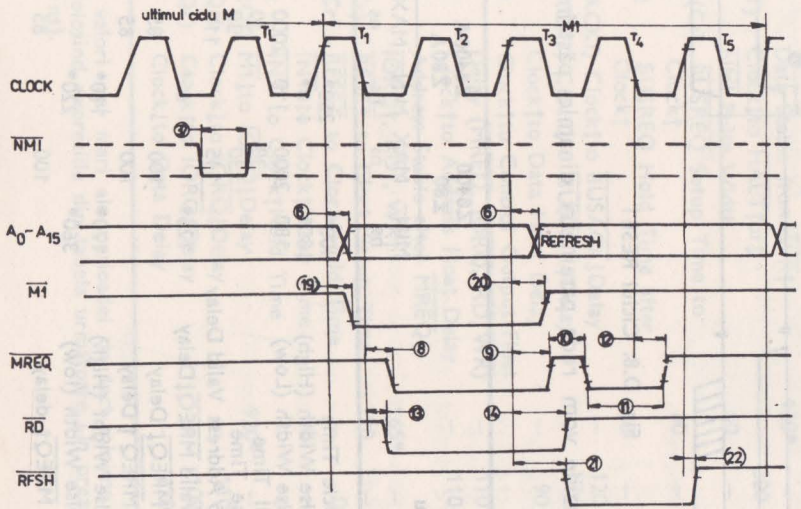


Fig. D.3. Ciclul de citire/scriere I/O



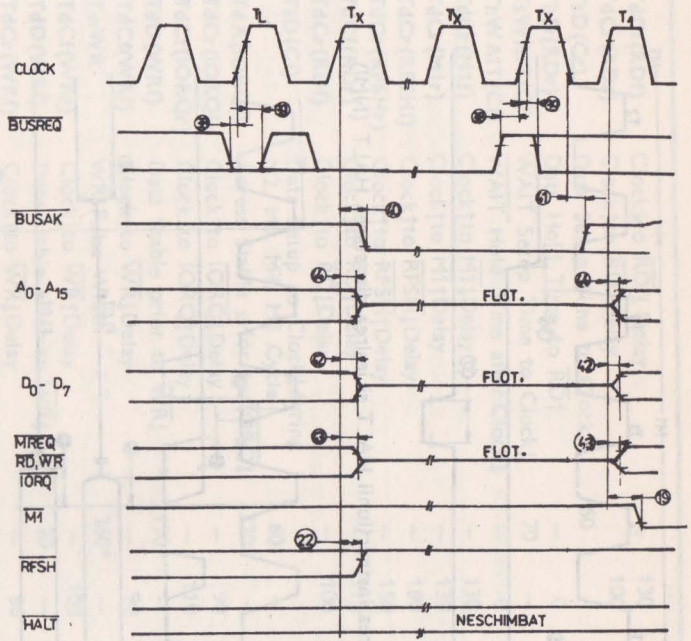
NOTA: - T_L - ultimul ciclu de tact al instructiunii precedente.
- T_w^* - doua stari WAIT inserate automat de procesor.

Fig. D.4. Ciclul de cerere/acceptare întrerupere



NOTA: -T_L - ultimul ciclu de tact al instructiunii precedente.
-Desi NMI este o intrare asincrona, frontul ei cazator trebuie sa apara, cel tirziu, odata cu frontul crescator al tactului T_L, pentru a fi acceptat in ciclul masina urmator.

Fig. D.5. Ciclul de întrerupere nemascabilă



NOTA: -T_L - ultimul ciclu de tact al unui ciclu masina.
-T_X - ciclu de tact oarecare.

Fig. D.6. Ciclul de cerere/acceptare cedare magistrale (BUSREQ ; BUSAK)

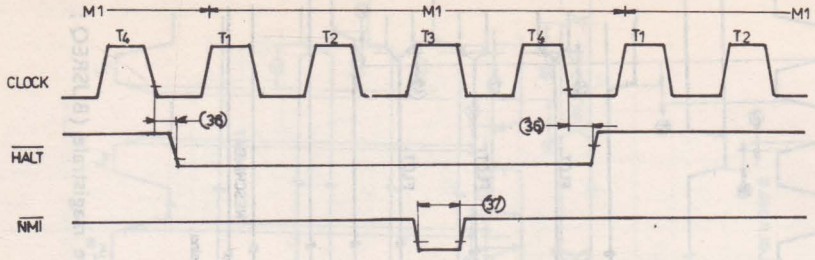


Fig. D.7. Acceptarea instrucţiunii HALT şi ieşirea din starea HALT (NMI)

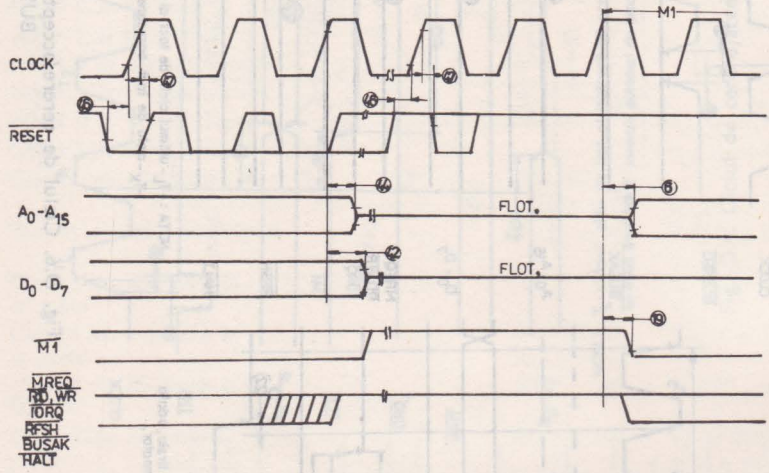


Fig. D.8. Ciclul RESET

Pentru a evita confuziile vom reda parametri dinamici păstrînd specificaţia lor originală.

Nr.	Simbol	Parametru	Z8400 Z80 MIN ns	Z8400 Z80 MAX ns	Z8400A Z80A MIN ns	Z8400A Z80A MAX ns	Z8400B Z80B MIN ns	Z8400B Z80B MAX ns
1.	TcC	Clock Cycle Time	400*		250*		165*	
2.	TwCh	Clock Pulse Width (High)	180*		110*		65*	
3.	TwCl	Clock Pulse Width (Low)	180	2000	110	2000	65	2000
4.	TfC	Clock Fall Time		30		30		20
5.	TrC	Clock Rise Time		30		30		20
6.	TdCr(A)	Clock↑ to Address Valid Delay		145		110		90
7.	TdA(MREQf)	Address Valid $\overline{\text{MREQ}}$↓Delay	125*		65*		35*	
8.	TdCf(MREQf)	Clock↓to $\overline{\text{MREQ}}$↓ Delay		100		85		70
9.	TdCr(MREQr)	Clock↑to $\overline{\text{MREQ}}$ ↑ Delay		100		85		70
10.	TwMREQh	$\overline{\text{MREQ}}$ Pulse Widht (High)	170*		110*		65*	
11.	TwMREQl	$\overline{\text{MREQ}}$ Pulse Width (low)	360*		220*		135*	
12.	TdCf(MREQr)	Clock↓to $\overline{\text{MREQ}}$↑ delay		100		85		70

13.	TdCf(RDf)	Clock↓to $\overline{RD}$↓ Delay		130		95		80
14.	TdCr(RDr)	Clock↑to $\overline{RD}$↑Delay		100		85		70
15.	TsD(Cr)	Data Setup Time to Clock↑	50		35		30	
16.	ThD(RDr)	Data Hold Time to $\overline{RD}$↑	—	0	—	0	—	
17.	TsWAIT(Cf)	$\overline{WAIT}$ Setup Time to Clock↓	70	—	70	—	60	—
18.	ThWAIT(Cf)	$\overline{WAIT}$ Hold Time after Clock↓	—	0	—	0	—	0
19.	TdCr(M1f)	Clock↑to $\overline{M1}$↓Delay	—	130	—	100	—	80
20.	TdCr(M1r)	Clock↑to $\overline{M1}$↑Delay	—	130	—	100	—	80
21.	TdCr(RFSHf)	Clock↑to $\overline{RFSH}$↓Delay	—	180	—	130	—	110
22.	TdCr(RFSHr)	Clock↑to $\overline{RFSH}$↑Delay	—	150	—	120	—	100
23.	TdCf(RDr)	Clock↓to $\overline{RD}$↑Delay	—	110	—	85	—	70
24.	TdCr(RDf)	Clock↑to $\overline{RD}$↓Delay	—	100	—	85	—	70
25.	TsD(Cf)	Data Setup to Clock↓during M2, M3, M4, M5 Cycle	60	—	50	—	40	—
26.	TdA(IORQf)	Adress Stable prior to $\overline{IORQ}$↓	320*	—	180*	—	110*	—
27.	TdCr(IORQf)	Clock↑to $\overline{IORQ}$↓Delay	—	90	—	75	—	65
28.	TdCf(IORQr)	Clock↓to $\overline{IORQ}$↑Delay	—	110	—	85	—	70
29.	TdD(WRf)	Data Stable prior to $\overline{WR}$↓	190*	—	80*	—	25*	—
30.	TdCf(WRf)	Clock↓to $\overline{WR}$↓Delay	—	90	—	80	—	20
31.	TwWR	$\overline{WR}$ Pulse Width	360*	—	220*	—	135*	—
32.	TdCf(WRr)	Clock↓to $\overline{WR}$↑Delay	—	100	—	80	—	70
33.	TdD(WRf)	Data stable prior to $\overline{WR}$↓	20	—	10	—	55	—
34.	TdCr(WRf)	Clock↑to $\overline{WR}$↓Delay	—	80	—	65	—	60
35.	TdWRr(D)	Data Stable from $\overline{WR}$↑	120*	—	60*	—	30*	—
36.	TdCf(HALT)	Clock↓to $\overline{HALT}$↑or↓	—	300	—	300	—	260
37.	TwNMI	$\overline{NMI}$ Pulse Width	80	—	80	0	70	—
38.	TsBUSREQ(Cr)	$\overline{BUSREQ}$ Setup Time to Clock↑	80	—	50	—	50	—
39.	ThBUSREQ (Cr)	BUSREQ Hold Time after Clock↑	0	—	0	—	0	—
40.	TdCr(BUSACKf)	Clock↑to $\overline{BUSACK}$↓Delay	—	120	—	100	—	90
41.	TdCf(BUSACKr)	Clock↓to $\overline{BUSACK}$↑Delay	—	110	—	100	—	90
42.	TdCr(Dz)	Clock↑to Data Float Delay	—	90	—	90	—	80
43.	TdCr(CTz)	Clock↑to Control Output Float Delay ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$)	—	110	—	90	—	80
44.	TdCr(Az)	Clock↑to Address Float Delay	—	110	—	90	—	80
45.	TdCTr(A)	Address Stable after $\overline{MREQ}$↑, $\overline{IORQ}$↑, $\overline{RD}$↑, $\overline{WR}$↑	165*	—	80*	—	35*	—
46.	TsRESET(Cr)	$\overline{RESET}$ to Clock↑Setup Time	90	—	60	—	60	—
47.	ThRESET(Cr)	$\overline{RESET}$ to Clock↑Hold Time	—	0	—	0	—	0
48.	TsINT(Cr)	$\overline{INT}$ to Clock↑Setup Time	80	—	80	—	70	—
49.	ThINT(Cr)	$\overline{INT}$ to Clock↑Hold Time	—	0	—	0	—	0
50.	TdM1f(IORQf)	$\overline{M1}$↓to $\overline{IORQ}$↓Delay	920*	—	565*	—	363*	—
51.	TdCf(IORQf)	Clock↓to $\overline{IORQ}$↓Delay	—	110	—	85	—	70
52.	TdCr(IORQr)	Clock↑to $\overline{IORQ}$↑Delay	—	100	—	85	—	70
53.	TdCf(D)	Clock↓to Data Delay	—	230	—	150	—	130

* Pentru valori mai mari ale perioadei de tact minime impuse, valorile normate se vor recalcula folosind expresiile din tabela următoare (în prezentele valori s-au considerat TrC = TfC = 20 ns).

Condiții de test folosite :

VIH = 2,0 V
VIL = 0,8 V
VIHC = V_{cc} − 0,6 V
VILC = 0,45 V

VOH = 2,0 V
VOL = 0,8 V

Calculul parametrilor care depind de frecvența de tact

Nr.	Simbol	Z8400 Z80	Z8400A Z80A	Z8400B Z80B
1.	TcC	TwCh + TwCl + TrC + TfC		
2.	TwCh	Deși regiștri microprocesorului sînt proiectați a fi statici nealterarea lor nu se garantează pentru TwCh mai mare decît 200 microsecunde.		
7.	TdA(MREQf)	TwCh+TfC−75	TwCh+TfC−65	TwCh+TfC−50
10.	TwMREQl	TwCh+TfC−30	TwCh+TfC−20	TwCh+TfC−30
11.	TwMREQl	TcC−40	TcC−30	TcC−30
26.	TdA(IORQf)	TcC−80	TcC−70	TcC−55
29.	TdD(WRf)	TcC−210	TcC−170	TcC−140
31.	TwWR	TcC−40	TcC−30	TcC−30
33.	TdD(WRf)	TwCl+TrC−180	TwCl+TrC−140	TwCl+TrC−140
35.	TdWRr(D)	TwCl+TrC−80	TwCl+TrC−70	TwCl+TrC−55
45.	TdCTr(A)	TwCl+TrC−40	TwCl+TrC−50	TwCl+TrC−50
50.	TdM1f(IORQf)	2TcC+TwCh+TfC−80	2TcC+TwCh+TfC−65	2TcC+TwCh+TfC−50

După cum rezultă din sintaxa simbolurilor, tipul parametrului este caracterizat de litere mici iar semnalele între care se măsoară se caracterizează cu majuscule.

C — Cycle — ciclu (perioadă).
d — delay — întîrziere
h — high — nivel 1
l — low — nivel 0
w — width — lățime
f — fall — scăzător
r — rise — crescător
s — setup — prestabilire (devansare)
h — hold — menținere
z — float — stare flotantă (tristate)

Proiectantul va asigura îndeplinirea următorilor parametri : **1,2,3,4,5,15,16, 17,18,25,37,38,39,46,47,48,49.** Dintre aceștia cei critici sînt : **1,2,3,4,5,15 17.**

Restul parametrilor, cei care nu s-au menționat în ultimele două secvențe, vor fi asigurați de către microprocesor.

E

ANALIZA SETULUI DE INSTRUCȚIUNI ALE MICROPROCESORULUI Z 80

În prezentul capitol redăm împărțirea detailată pe clase și pe grupe de instrucțiuni.

Împărțirea pe grupe de instrucțiuni am realizat-o după cum urmează :

a) Am definit un set de date (informații) semnificative, care sînt în măsură să caracterizeze oricare instrucțiune procesor.

b) Am împărțit fiecare clasă în grupe de instrucțiuni, astfel încît membrii unei grupe să se asemene la majoritatea elementelor caracteristice definite.

c) Instrucțiunilor cuprinse în aceeași grupă le-am acordat mnemonici generice.

Redăm în cele ce urmează clasificarea propusă.

E.1. Clase și grupe de instrucțiuni.

Cl. 1. Instrucțiuni de transfer de 8 bit LOAD—8

Mnemonici incluse : LD

Totalul instrucțiunilor din clasă : 111

Numărul de grupe : 11

Grupa 1 :	**Instrucțiuni de transfer între regiștri :**	**LD r1,r2**
	Total instrucțiuni :	**49**
Grupa 2 :	**Instrucțiuni de încărcare regiștri din memorie :**	**LD r,(mem)**
	Total instrucțiuni :	**21**
Grupa 3 :	**Instrucțiuni de transfer conținut regiștri în memorie :**	**LD (mem),r**
	Total instrucțiuni :	**21**
Grupa 4 :	**Instrucțiuni de încărcare imediată a regiștrilor :**	**LD r,n**
	Total instrucțiuni :	**7**
Grupa 5 :	**Instrucțiuni de încărcare imediată a unor celule de memorie**	**LD(mem),n**
	Total instrucțiuni :	**3**

Grupa 6 :	Instrucţiuni de încărcare a acumulatorului din memorie :	LD A,(rr)
	Total instrucţiuni :	2
Grupa 7 :	Instrucţiuni de salvare a acumulatorului în memorie :	LD (rr), A
	Total instrucţiuni :	2
Grupa 8 :	Încărcarea acumulatorului din memorie prin adresare directă :	LD A,(nn)
	Total instrucţiuni :	1
Grupa 9 :	Salvarea acumulatorului în memorie prin adresare directă :	LD (nn),A
	Total instrucţiuni:	1
Grupa 10 :	Instrucţiuni de citire a regiştrilor hardware I şi R :	LD A,rh
	Total instrucţiuni :	2
Grupa 11 :	Instrucţiuni de programare a regiştrilor hardware I şi R :	LD rh,A
	Total instrucţiuni :	2

Cl. 2. Instrucţiuni de transfer de 16 bit — LOAD—16

Mnemonici incluse : LD,EX,EXX,POP,PUSH

Totalul instrucţiunilor din clasă : 39

Numărul de grupe : 10

Grupa 1 :	Instrucţiuni de încărcare imediată a regiştrilor dubli :	LD rr,nn
	Total instrucţiuni :	6
Grupa 2 :	Instrucţiuni de încărcare din memorie a regiştrilor dubli :	LD rr,(nn)
	Total instrucţiuni :	6
Grupa 3 :	Instrucţiuni de salvare în memorie a regiştrilor dubli :	LD (nn),rr
	Total instrucţiuni :	6
Grupa 4 :	Instrucţiuni de încărcare a indicatorului de stivă SP din regiştri dubli :	LD SP,rr
	Total instrucţiuni :	3
Grupa 5 :	Instrucţiuni de salvare pe stivă a regiştrilor dubli :	PUSH rr
	Total instrucţiuni :	6
Grupa 6 :	Instrucţiuni de restaurare a regiştrilor dubli din stivă :	POP rr
	Total instrucţiuni :	6
Grupa 7 :	Schimb de conţinut între regiştri dubli :	EX DE,HL
	Total instrucţiuni :	1
Grupa 8 :	Interschimbarea regiştrilor primari cu cei secundari :	EXX
	Total instrucţiuni :	1
Grupa 9 :	Interschimbarea regiştrilor de stare primari şi secundari :	EX AF,AF'
	Total instrucţiuni :	1
Grupa 10 :	Instrucţiuni de interschimbare a regiştrilor dubli cu vîrful stivei :	EX (SP),rr
	Total instrucţiuni :	3

Cl. 3. Instrucţiuni de transfer de blocuri de date : — LOAD—IDR

Mnemonici incluse : LDD, LDDR, LDI, LDIR

Totalul instrucţiunilor din clasă : 4

Numărul de grupe : 2

Grupa 1 :	Transferuri simple :	LDx
	Total instrucţiuni :	2
Grupa 2 :	Transferuri multiple :	LDxR
	Total instrucţiuni :	2

Cl. 4. Instrucţiuni aritmetice/logice pe 8 bit — AR/LOG—8

Mnemonici incluse : ADD,ADC,SUB,SBC,AND,XOR,OR, CP,INC,DEC,CPL,NEG,DAA,RLCA, RRCA, RLA,RRA

Totalul instrucţiunilor din clasă : 115
Numărul de grupe : 26

Grupa 1 :	**Aritmetice—Adunarea simplă a regiştrilor :** Total instrucţiuni :	**ADD A,r** 7
Grupa 2 :	**Aritmetice—Adunarea regiştrilor cu adăugarea transportului :** Total instrucţiuni :	**ADC A,r** 7
Grupa 3 :	**Aritmetice—Scăderea simplă a regiştrilor :** Total instrucţiuni :	**SUB r** 7
Grupa 4 :	**Aritmetice — Scăderea regiştrilor cu scăderea transportului :** Total instrucţiuni :	**SBC A,r** 7
Grupa 5 :	**Logice — Operaţia ŞI cu regiştri :** Total instrucţiuni :	**AND r** 7
Grupa 6 :	**Logice — Operaţia SAU EXCLUSIV cu regiştri :** Total instrucţiuni :	**XOR r** 7
Grupa 7 :	**Logice — Operaţia SAU cu regiştri :** Total instrucţiuni :	**OR r** 7
Grupa 8 :	**Logice — Comparaţia acumulatorului cu regiştri :** Total instrucţiuni :	**CP r** 7
Grupa 9 :	**Aritmetice — Adunarea simplă cu locaţii de memorie :** Total instrucţiuni :	**ADD A,(mem)** 3
Grupa 10 :	**Aritmetice — Adunarea cu adăugarea transportului (memorie) :** Total instrucţiuni :	**ADC A,(mem)** 3
Grupa 11 :	**Aritmetice — Scăderea simplă cu locaţii de memorie :** Total instrucţiuni :	**SUB (mem)** 3
Grupa 12 :	**Aritmetice — Scăderea cu scăderea transportului (memorie) :** Total instrucţiuni :	**SBC A,(mem)** 3
Grupa 13 :	**Logice — Operaţia SI cu locaţii de memorie :** Total instrucţiuni :	**AND (mem)** 3
Grupa 14 :	**Logice — Operaţia SAU EXCLUSIV cu locaţii de memorie :** Total instrucţiuni :	**XOR (mem)** 3
Grupa 15 :	**Logice — Operaţia SAU cu locaţii de memorie :** Total instrucţiuni :	**OR (mem)** 3
Grupa 16 :	**Logice — Comparaţia acumulatorului cu locaţii de memorie :** Total instrucţiuni :	**CP (mem)** 3
Grupa 17 :	**Instrucţiuni aritmetice şi logice imediate** **ADD A,n ; ADC A,n ; SUB n ; SBC A,n ; AND n ; XOR n ; OR n ; CP n ;** Total instrucţiuni	 8
Grupa 18 :	**Incrementarea regiştrilor simpli :** Total instrucţiuni :	**INC r** 7
Grupa 19 :	**Decrementarea regiştrilor simpli :** Total instrucţiuni :	**DEC r** 7
Grupa 20 :	**Incrementarea locaţiilor de memorie :** Total instrucţiuni :	**INC (mem)** 3
Grupa 21 :	**Decrementarea locaţiilor de memorie :** Total instrucţiuni :	**DEC (mem)** 3
Grupa 22 :	**Complementul faţă de 1 :** Total instrucţiuni :	**CPL** 1
Grupa 23 :	**Complementul faţă de 2 :** Total instrucţiuni :	**NEG** 1

Grupa 24 :	**Ajustarea BCD a acumulatorului :**	**DAA**
	Total instrucţiuni :	1
Grupa 25 :	**Rotirea acumulatorului „cu" carry :**	**RxCA**
	Total instrucţiuni :	2
Grupa 26 :	**Rotirea acumulatorului „prin" carry :**	**RxA**
	Total instrucţiuni :	2

Cl. 5. Instrucţiuni aritmetice pe 16 bit — ARIT—16

Mnemonici incluse : ADD,ADC,SBC,INC,DEC
Totalul instrucţiunilor din clasă : 32
Numărul de grupe : 7

Grupa 1 :	**Adunarea simplă cu acumulatorul în HL :**	**ADD HL,rr**
	Total instrucţiuni :	4
Grupa 2 :	**Adunarea simplă cu acumulatorul în IX :**	**ADD IX,rr**
	Total instrucţiuni :	4
Grupa 3 :	**Adunarea simplă cu acumulatorul în IY :**	**ADD IY,rr**
	Total instrucţiuni :	4
Grupa 4 :	**Adunare cu adunarea transportului (carry) :**	**ADC HL,rr**
	Total instrucţiuni :	4
Grupa 5 :	**Scădere cu scăderea transportului (carry) :**	**SBC HL,rr**
	Total instrucţiuni :	4
Grupa 6 :	**Incrementarea regiştrilor dubli :**	**INC rr**
	Total instrucţiuni :	6
Grupa 7 :	**Decrementarea regiştrilor dubli :**	**DEC rr**
	Total instrucţiuni :	6

Cl. 6. Instrucţiuni logice pe blocuri de date — LOG—IDR

Mnemonici incluse : CPD,CPDR,CPI,CPIR
Totalul instrucţiunilor din clasă : 4
Numărul de grupe : 2

Grupa 1 :	**Comparări simple :**	**CPx**
	Total instrucţiuni :	2
Grupa 2 :	**Comparări multiple :**	**CPxR**
	Total instrucţiuni :	2

Cl. 7. Instrucţiuni orientate pe bit — BITSR

Mnemonici incluse : BIT,SET,RES,SCF,CCF
Totalul instrucţiunilor din clasă : 242
Numărul de grupe 7

Grupa 1 :	**Testarea biţilor din regiştri :**	**BIT x,r**
	Total instrucţiuni :	56
Grupa 2 :	**Înscrierea (set) biţilor din regiştri :**	**SET x,r**
	Total instrucţiuni :	56
Grupa 3 :	**Ştergerea (reset) biţilor din regiştri :**	**RES x,r**
	Total instrucţiuni :	56
Grupa 4 :	**Testarea biţilor din locaţii de memorie :**	**BIT x,(mem)**
	Total instrucţiuni :	24
Grupa 5 :	**Înscrierea (set) biţilor din locaţii de memorie :**	**SET x,(mem)**
	Total instrucţiuni :	24

Grupa 6 :	**Ştergerea (reset) biţilor din locaţii de memorie :**	**RES x, (mem)**
	Total instrucţiuni :	24
Grupa 7 :	**Manipularea bitului de transport (SCF,CCF) :**	**xCF**
	Total instrucţiuni :	2

Cl. 8. Instrucţiuni de salt — JUMP

Mnemonici incluse : JP,JR,DJNZ
Totalul instrucţiunilor din clasă : 18
Numărul de grupe : 6

Grupa 1 :	**Salt necondiţionat absolut (direct) :**	**JP nn**
	Total instrucţiuni :	1
Grupa 2 :	**Salt necondiţionat absolut (indirect) :**	**JP (rr)**
	Total instrucţiuni :	3
Grupa 3 :	**Salt condiţionat absolut (direct)**	**JP c,nn**
	Total instrucţiuni :	8
Grupa 4 :	**Salt necondiţionat relativ :**	**JR d**
	Total instrucţiuni :	1
Grupa 5 :	**Salt condiţionat relativ :**	**JR c,d**
	Total instrucţiuni :	4
Grupa 6 :	**Salt condiţionat relativ, cu decrementare :**	**DJNZ d**
	Total instrucţiuni :	1

Cl. 9. Instrucţiuni de apel şi revenire din subrutine — CALL/RET

Mnemonici incluse : CALL, RET, RST
Totalul instrucţiunilor din clasă : 26
Numărul de grupe : 5

Grupa 1 :	**Apel necondiţionat :**	**CALL nn**
	Total instrucţiuni :	1
Grupa 2 :	**Revenire necondiţionată :**	**RET**
	Total instrucţiuni :	1
Grupa 3 :	**Apel condiţionat :**	**CALL c,nn**
	Total instrucţiuni :	8
Grupa 4 :	**Revenire condiţionată :**	**RET c**
	Total instrucţiuni :	8
Grupa 5 :	**Apel scurt, necondiţionat, la adrese fixe :**	**RST n**
	Total instrucţiuni :	8

Cl. 10. Instrucţiuni de rotire şi deplasare — ROT/SHIFT

Mnemonici incluse : RLC,RRC,RL,RR,SLA,SRA,SRL,RID,RRD
Totalul instrucţiunilor din clasă : 72
Numărul de grupe : 9

Grupa 1 :	**Rotiri „cu" carry a regiştrilor (RLC,RRC) :**	**RxC r**
	Total instrucţiuni :	14
Grupa 2 :	**Rotiri „prin" carry a regiştrilor (RL,RR) :**	**Rx r**
	Total instrucţiuni :	14
Grupa 3 :	**Deplasări (shift) aritmetice a regiştrilor (SLA,SRA) :**	**SxA r**
	Total instrucţiuni :	14
Grupa 4 :	**Deplasări (shift) logice a regiştrilor :**	**SRL r**
	Total instrucţiuni :	7

Grupa 5 :	**Rotiri „cu" carry a locațiilor de memorie (RLC,RRC) :**	**RxC (mem)**
	Total instrucțiuni :	6
Grupa 6 :	**Rotiri „prin" carry a locațiilor de memorie (RL,RR) :**	**Rx (mem)**
	Total instrucțiuni :	6
Grupa 7 :	**Deplasări (shift) aritmetice a locațiilor de memorie (SLA, SRA) :**	**SxA (mem)**
	Total instrucțiuni :	6
Grupa 8 :	**Deplasări (shift) logice a locațiilor de memorie :**	**SRL (mem)**
	Total instrucțiuni :	3
Grupa 9 :	**Rotirea digiților de 4 biți (RLD,RRD) :**	**RxD**
	Total instrucțiuni :	2

Cl. 11. Instrucțiuni de intrare/ieșire — IN/OUT

Mnemonici incluse : IN,IND,INDR,INI,INIR,OTDR,OTIR, OUT,OUTD,OUTI

Totalul instrucțiunilor din clasă : 24

Numărul de grupe : 8

Grupa 1 :	**Input cu adresare directă :**	**IN A,(n)**
	Total instrucțiuni :	1
Grupa 2 :	**Input cu adresare indirectă :**	**IN r,(C)**
	Total instrucțiuni :	7
Grupa 3 :	**Output cu adresare directă :**	**OUT (n), A**
	Total instrucțiuni :	1
Grupa 4 :	**Output cu adresare indirectă :**	**OUT (C), r**
	Total instrucțiuni :	7
Grupa 5 :	**I/E blocuri de date — input simplu (IND,INI) :**	**INx**
	Total instrucțiuni :	2
Grupa 6 :	**I/E blocuri de date — input multiplu (INDR,INIR) :**	**INxR**
	Total instrucțiuni :	2
Grupa 7 :	**I/E blocuri de date — output simplu (OUTD,OUTI) :**	**OUTx**
	Total instrucțiuni :	2
Grupa 8 :	**I/E blocuri de date — output multiplu (OTDR,OTIR) :**	**OTxR**
	Total instrucțiuni :	2

Cl. 12 Instrucțiuni de comandă — SYS

Mnemonici incluse : DI,EI,HALT,IM,NOP,RETI,RETN

Totalul instrucțiunilor din clasă : 9

Numărul de grupe : 5

Grupa 1 :	**Instrucțiunea HALT :**	**HALT**
	Total instrucțiuni :	1
Grupa 2 :	**Stabilirea modului de întrerupere :**	**IM x**
	Total instrucțiuni :	3
Grupa 3 :	**Validarea și inhibarea sistemului de întreruperi (EI,DI) :**	**xI**
	Total instrucțiuni :	2
Grupa 4 :	**Revenire din rutinele de tratare a întreruperilor (RETI,RETN) :**	**RETx**
	Total instrucțiuni :	2
Grupa 5 :	**Instrucțiunea NOP :**	**NOP**
	Total instrucțiuni :	1

Iată datele (informațiile) caracteristice menite să definească complet oricare instrucțiune mașină a microprocesorului Z80 :

1. Mnemonică
2. Acțiunea
3. Necesarul de memorie
4. Necesarul de timp (timp de execuție)
5. Structura codului
6. Tipuri de adresare folosite
7. Flag-uri afectate.

Pe paginile următoare găsiți cîte o filă (una sau două pagini) pentru fiecare din cele 98 de grupe de instrucțiuni enumerate la cap. E.1.

Clasa 1 : Instrucțiuni de transfer de 8 bit

Grupa 1 : Instrucțiuni de transfer între regiștri

Mnemonica generică : **LD r1,r2**

unde : r1 și r2 pot fi : A, B, C, D, E H sau L

Total instrucțiuni : 7 * 7 = 49

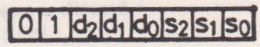
Acțiunea : **r1 ← r2**

Conținutul registrului **r2** este transferat în registrul **r1**. Vechiul conținut a lui **r1** se pierde. Conținutul lui **r2** rămîne neschimbat.

Necesar de memorie : 1 byte

Necesar de timp : 4(4) tacți procesor

Structura instrucțiunii



byte

unde : **d2d1d0** reprezintă codul pe 3 bit al registrului destinație **r1**
s2s1s0 reprezintă codul pe 3 bit al registrului sursă **r2**

codurile posibile sînt :

000	— B	100	— H
001	— C	101	— L
010	— D	110	
011	— E	111	— A

Tipuri de adresare : **destinație / sursă**
implicit / implicit

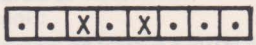
Memento de cod : **LD r1,r2**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
LD B,r 40H+	B,B	B,C	B,D	B,E	B,H	B,L		B,A	C,B	C,C	C,D	C,E	C,H	C,L		C,A	40H+LD C,r
LD D,r 50H+	D,B	D,C	D,D	D,E	D,H	D,L		D,A	E,B	E,C	E,D	E,E	E,H	E,L		E,A	50H+LD E,r
LD H,r 60H+	H,B	H,C	H,D	H,E	H,H	H,L		H,A	L,B	L,C	L,D	L,E	L,H	L,L		L,A	60H+LD L,r
70H+									A,B	A,C	A,D	A,E	A,H	A,L		A,A	70H+LD A,r

Codul dorit se obține însumînd cele două numere aferente rîndului și coloanei instrucțiunii căutate.

Exemplu : LD D,L 50H + 5 = 55H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II p. 144

Clasa 1 : Instrucţiuni de transfer de 8 bit
Grupa 2 : Instrucţiuni de încărcare regiştri din memorie (read memory)

Mnemonica generică : **LD r,(mem)**

unde : r poate fi A, B, C, D, E, H sau L
mem poate fi HL, IX+IND sau IY+IND

Total instrucţiuni : 3 * 7 = 21

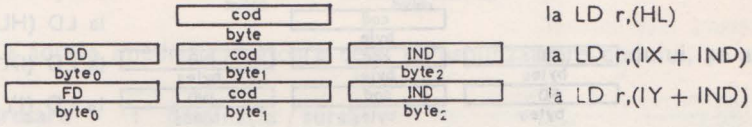
Acţiunea : r ⟵ (mem)

Conţinutul celulei de memorie avînd adresa specificată prin **mem** este transferat în registrul **r**.
Vechiul conţinut al lui **r** se pierde. Conţinutul celulei de memorie folosite rămîne neschimbat.

Necesar de memorie : 1 byte la LD r.(HL)
3 byte la LD r,(IX+IND) sau
LD r,(IY+IND)

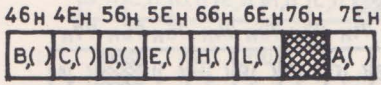
Necesar de timp : 7 (4, 3) tacţi procesor la LD r, (HL)
19 (4, 4, 3, 5, 3) tacţi procesor la
LD r,(IX+IND) sau
LD r,(IY+IND)

Structura instrucţiunii :



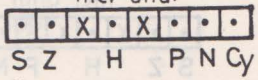
Tipuri de adresare : destinaţie / sursă
implicit / indirect la LD r,(HL)
implicit / indexat la LD r,(IX+IND)
sau LD r,(IY+IND)

Memento de cod : **LD r,(mem)**



Codul obţinut este acelaşi pentru toate cele trei tipuri de instrucţiuni. La instrucţiunile indexate el apare în byte 1.

Flaguri afectate : nici unul



Exemple de utilizare : vol. II p. 168

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 3 : Instrucţiuni de transfer conţinut regiştri în memorie (write memory)

Mnemonica generică : LD (mem),r

unde : **mem** poate fi **HL, IX+IND,** sau **IY+IND**
r poate fi **A, B, C, D, E, H** sau **L**

Total instrucţiuni : $3 * 7 = 21$

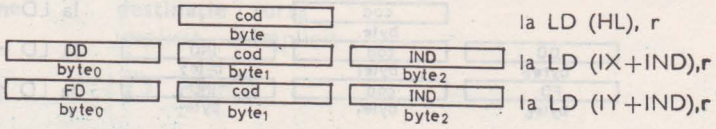
Acţiunea : **(mem) ⟵ r**

Conţinutul registrului **r** este transferat în memorie la adresa specificată prin **mem.** Conţinutul lui **r** rămîne neafectat. Vechiul conţinut al celulei de memorie în care s-a făcut transferul se pierde.

Necesar de memorie : 1 byte la LD (HL), r
3 byte la LD (IX+IND), r sau
LD (IY+IND), r

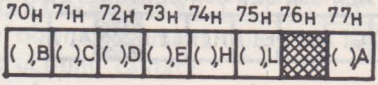
Necesar de timp : 7 (4, 3) tacţi procesor la LD (HL), r
19 (4, 4, 3, 5, 3) tacţi procesor la
LD (IX+IND), r sau
LD (IY+IND), r

Structura instrucţiunii :



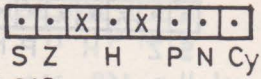
Tipuri de adresare : **destinaţie / sursă**
indirect / implicit la LD (HL), r
indexat / implicit la LD (IX+IND), r
sau LD (IY+IND), r

Memento de cod : **LD (mem),r**



Codul obţinut este acelaşi pentru toate cele trei tipuri de instrucţiuni. La instrucţiunile indexate el apare în byte1.

Flaguri afectate : nici unul



Exemple de utilizare : **vol II 210**

Clasa 1 : Instrucţiuni de transfer de 8 bit
Grupa 4 : Instrucţiuni de încărcare imediată a regiştrilor

Mnemonica generică : **LD r,n**

unde : r poate fi A, B, C, D, E, H sau L
n este un număr [0,255]

Total instrucţiuni : 7

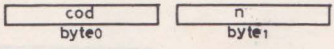
Acţiunea : r ⟵ n

Numărul n este înscris în registrul r. Vechea valoare cuprinsă în r se pierde.

Necesar de memorie : 2 byte

Necesar de timp : 7 (4,3) tacţi procesor

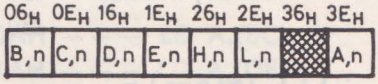
Structura instrucţiunii :



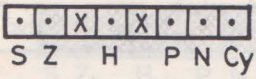
La adresa imediat următoare celei corespunzătoare codului, se află **n**

Tipuri de adresare : **destinaţie / sursă**
implicit / imediat

Memento de cod : **LD r,n**



Flaguri afectate : nici unul



Exemple de utilizare : **vol. II 166**

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 5 : Instrucţiuni de încărcare imediată a unor celule de memorie (write memory)

Mnemonica generică : **LD (mem),n**

unde : **mem** poate fi **HL, IX+IND** sau **IY+IND**
n este un număr **[0,255]**

Total instrucţiuni : 3 * 1 = 3

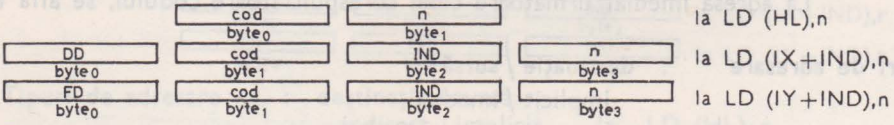
Acţiunea : (mem) ⟵ n

Numărul **n** este înscris în celula de memorie a cărei adresă este specificată prin **mem**. Vechiul conţinut al acelei celule de memorie se pierde.

Necesar de memorie : 2 byte la LD (HL), n
4 byte la LD (IX+IND),n sau
la LD (IY+IND),n

Necesar de timp : 10 (4,3,3) la LD (HL),n
19 (4,4,3,5,3) la LD (IX+IND),n sau
la LD (IY+IND),n

Structura instrucţiunii :

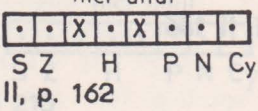


Tipuri de adresare : **destinaţie / sursă**

indirect / imediat la LD (HL),n
indexat / imediat la LD (IX+IND),n
sau LD (IY+IND),n

Memento de cod : **LD (mem),n**

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 162**

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 6 : Instrucţiuni de încărcare a acumulatorului din memorie (read memory)

Mnemonica generică : **LD A,(rr)**

unde : rr poate fi BC sau DE

Total instrucţiuni : 1 * 2 = 2

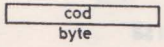
Acţiunea : A ⟵ (rr)

Conţinutul locaţiei de memorie adresată prin registrul dublu rr, este transferat în acumulator. Conţinutul vechi al acumulatorului se pierde Conţinutul celulei de memorie implicate rămîne neschimbat.

Necesar de memorie : 1 byte

Necesar de timp : 7 (4,3) tacţi procesor

Structura instrucţiunii :

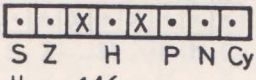


Tipuri de adresare : destinaţie / sursă
implicit / indirect

Memento de c› : LD A,(rr)

: 0AH — — — LD A,(BC)
1AH — — — LD A,(DE)

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 146

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 7 : Instrucţiuni de salvare a acumulatorului în memorie (write memory)

Mnemonica generică : **LD (rr),A**

unde : rr poate fi BC sau DE

Total instrucţiuni : 1 * 2 = 2

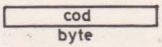
Acţiunea : (rr) ⟵ A

Conţinutul acumulatorului este înscris în celula de memorie având adresa specificată în registrul dublu rr. Vechiul conţinut al celulei de memorie se pierde. Conţinutul acumulatorului rămîne neschimbat.

Necesar de memorie : 1 byte

Necesar de timp : 7 (4,3) tacţi procesor

Structura instrucţiunii :

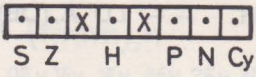


Tipuri de adresare : destinaţie / sursă
indirect / implicit

Memento de cod : LD (rr),A

02H ——— LD (BC),A
12H ——— LD (DE),A

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 146

Clasa 1 : Instrucţiuni de transfer de 8 bit.

Grupa 8 : Încărcarea acumulatorului din memorie, prin adresare directă.

Mnemonica generică : **LD A,(nn)**

unde : nn este o adresă [0,65535]

Total instrucţiuni : 1

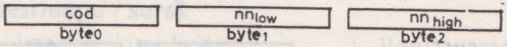
Acţiunea : A ⟵ (nn)

Conţinutul celulei de memorie avînd adresa **nn** este transferat în acumulator. Vechiul conţinut al acumulatorului se pierde. Conţinutul celulei de memorie folosite nu se schimbă.

Necesar de memorie : 3 byte

Necesar de timp : 13 (4,3,3,3) tacţi procesor

Structura instrucţiunii :



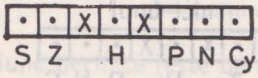
byte 1 conţine octetul inferior al adresei nn
byte 2 conţine octetul superior al adresei nn

Tipuri de adresare : **destinaţie / sursă**
implicit / direct

Memento de cod : **LD A,(nn)**

cod : 3AH

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 210

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 9 : Salvarea acumulatorului în memorie prin adresare directă

Mnemonica generică : **LD (nn),A**

unde : nn este o adresă [0,65535]

Total instrucţiuni : 1

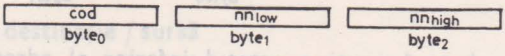
Acţiunea : (nn) ⟵ A

Conţinutul acumulatorului este depus în celula de memorie care are adresa nn. Vechiul conţinut al celulei respective se pierde. Conţinutul acumulatorului rămîne neschimbat.

Necesar de memorie : 3 byte

Necesar de timp : 13 (4,3,3,3) tacţi procesor

Structura instrucţiunii :



byte 1 conţine octetul inferior al adresei **nn**

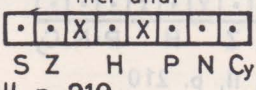
byte 2 conţine octetul superior al adresei **nn**

Tipuri de adresare : **destinaţie / sursă**
direct / implicit

Memento de cod : LD (nn),A

cod : **32H**

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 210

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 10 : Instrucţiuni de citire a regiştrilor hardware I şi R

Mnemonica generică : **LD A,rh**

unde : **rh** poate fi **I** sau **R**

Total instrucţiuni : 1 + 1 = 2

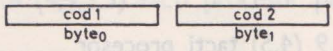
Acţiunea : A ← rh

Conţinutul registrului special **I** sau **R** este transferat în **A**. Vechiul conţinut al registrului **A** se pierde. Conţinutul registrului hard nu se schimbă.

Necesar de memorie : 2 byte

Necesar de timp : 9 (4,5) tacţi procesor

Structura instrucţiunii :



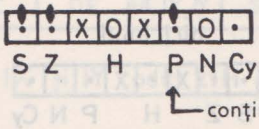
Se execută două cicluri fetch (M1) consecutive.

Tipuri de adresare : **destinaţie / sursă**
implicit / implicit

Memento de cod : LD A,rh

EDH 57H ——— LD A,I
EDH 5FH ——— LD A,R

Flaguri afectate : unica grupă de transfer de 8 bit, care afectează indicatorii de condiţie : sînt afectaţi toţi indicatorii cu excepţia lui carry.



Această instrucţiune permite (este de altfel unica soluţie) identificarea stării sistemului de întreruperi pe cale software.
Aspectul este interesant în sisteme cu întreruperi multiple.

Exemple de utilizare : **vol. II, p. 197**

Clasa 1 : Instrucţiuni de transfer de 8 bit

Grupa 11 : Instrucţiuni de programare a regiştrilor hardware I şi R

Mnemonica generică : **LD rh,A**

unde : **rh** poate fi **I** sau **R**

Total instrucţiuni : 1 + 1 = 2

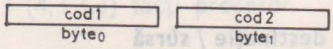
Acţiunea : **rh ⟵ A**

Conţinutul acumulatorului este transferat în registrul special selectat. Vechiul conţinut al acestui registru se pierde. Conţinutul acumulatorului nu se schimbă.

Necesar de memorie : 2 byte

Necesar de timp : 9 (4,5) tacţi procesor

Structura instrucţiunii :



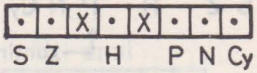
Se execută două cicluri fetch (M1) consecutive.

Tipuri de adresare : **destinaţie / sursă**
implicit / implicit

Memento de cod : **LD rh,A**

EDH 47H — — — LD I,A
EDH 4FH — — — LD R,A

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 146

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 1 : Instrucţiuni de încărcare imediată a registrilor dubli

Mnemonica generică : **LD rr,nn**

unde : rr poate fi BC, DE, HL, SP, IX sau IY
nn este un număr [0,65535]

Total instrucţiuni : $6 * 1 = 6$

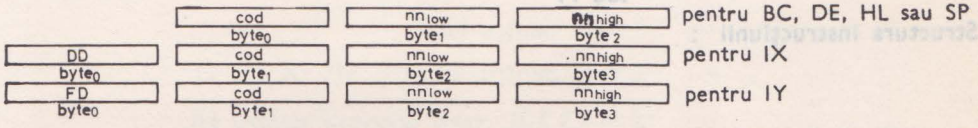
Acţiunea : rrlow ⟵ nnlow
rrhigh ⟵ nnhigh

Registrul dublu **rr** este încărcat cu valoarea **nn**. Vechiul conţinut al registrului dublu se pierde.

Necesar de memorie : 3 byte pentru BC, DE, HL sau SP
4 byte pentru IX sau IY

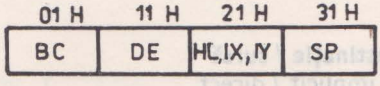
Necesar de timp : 10 (4,3,3) tacţi procesor pentru BC, DE, HL sau SP
14 (4,4,3,3) tacţi procesor pentru IX sau IY

Structura instrucţiunii :



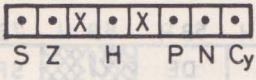
Tipuri de adresare : **destinaţie / sursă**
implicit / imediat

Memento de cod : **LD rr,nn**



Pentru regiştri IX şi IY codul **21H** este precedat de **DDH** respectiv **FDH**

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 114**

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 2 : Instrucţiuni de încărcare din memorie a regiştrilor dubli (read memory)

Mnemonica generică : **LD rr,(nn)**

unde : **rr** poate fi **BC, DE, HL, SP, IX** sau **IY**
nn este o adresă **[0,65535]**

Total instrucţiuni : $1 + 2 + 3 = 6$

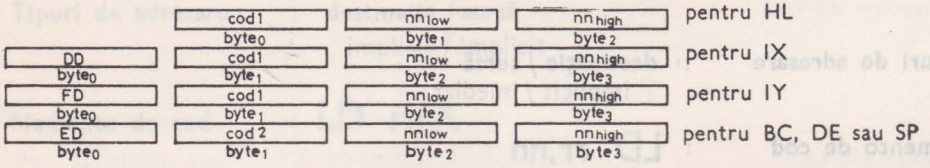
Acţiunea : $rr_{low} \longleftarrow (nn)$
$rr_{high} \longleftarrow (nn + 1)$

Registrul dublu **rr** este încărcat din memorie, cu 2 octeţi, începînd de la adresa **nn**. Conţinutul vechi al registrului dublu se pierde. Conţinutul celulei de memorie implicate rămîne neschimbat.

Necesar de memorie : 3 byte pentru HL
4 byte pentru BC, DE, SP, IX sau IY

Necesar de timp : 16 (4,3,3,3,3) tacţi procesor pentru HL
20 (4,4,3,3,3,3) tacţi procesor pentru BC, DE, SP, IX sau IY

Structura instrucţiunii :

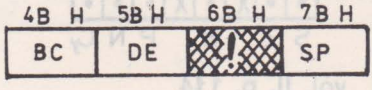


Tipuri de adresare : **destinaţie / sursă**
implicit / direct

Memento de cod : **LD rr,(nn)**

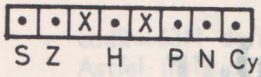
cod1 : **2AH** pentru **HL, IX, IY**

cod2 :



Observație : **cod2 = 6BH** definește aceeași instrucțiune ca și **cod1 = 2AH** ; astfel instrucțiunea **LD HL,(nn)** este definită pentru două secvențe de cod :
2A nn low nnhigh și **ED 6B nnlow nnhigh**

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 149

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 3 : Instrucţiuni de salvare în memorie a regiştrilor dubli (write memory) prin adresare directă

Mnemonica generică : **LD (nn),rr**

unde : **nn** este o adresă **[0,65535]**
r r poate fi **BC, DE, HL, SP, IX** sau **IY**

Total instrucţiuni : 1 + 2 + 3 = 6

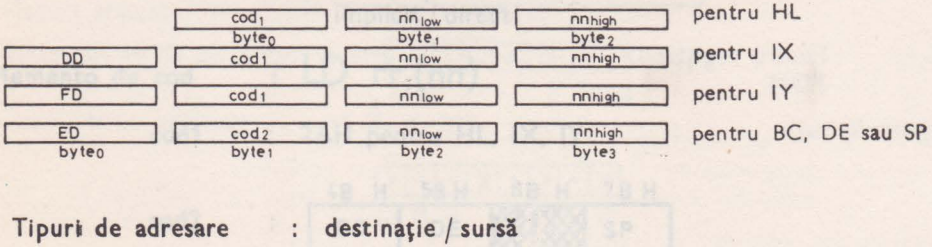
Acţiunea : **(nn) ⟵ rrlow**
(nn + 1) ⟵ rrhigh

Registrul dublu **rr** este transferat în memorie în două locaţii succesive, începînd cu adresa **nn**. Vechiul conţinut al celulelor specificate se pierde. Conţinutul registrului dublu rămîne neafectat.

Necesar de memorie : 3 byte pentru HL
: 4 byte pentru BC, DE, SP, IX sau IY

Necesar de timp : 16 (4,3,3,3,3) tacţi procesor pentru HL
: 20 (4,4,3,3,3,3) tacţi procesor pentru BC, DE, SP, IX sau IY

Structura instrucţiunii :



Tipuri de adresare : **destinaţie / sursă**
direct / implicit

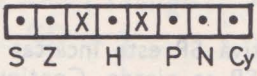
Memento de cod : LD (nn),rr

cod1 : 22H pentru HL,IX,IY

cod2 :

Observație : cod2=63H definește aceeași instrucțiune ca și 22H. Astfel instrucțiunea LD (nn),HL este definită pentru două secvențe de cod : 22 nnlow nnhigh și ED 63 nnlow nnhigh.

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 148

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 4 : Instrucţiuni de încărcare a indicatorului de stivă SP din regiştri dubli

Mnemonica generică : **LD SP,rr**

unde rr poate fi HL, IX sau IY

Total instrucţiuni : 3 * 1 = 3

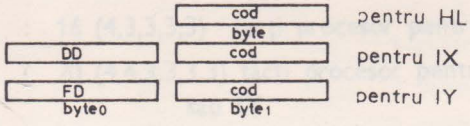
Acţiunea : SP ← rr

Indicatorul de stivă **SP** este încărcat din registrul dublu **rr**. Vechiul conţinut al lui **SP** se pierde. Conţinutul registrului sursă **rr** rămîne nealterat.

Necesar de memorie : 1 byte pentru HL
2 byte pentru IX sau IY

Necesar de timp : 6 (6) pentru HL
10(4,6) pentru IX sau IY

Structura instrucţiunii :

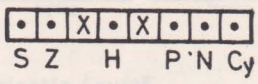


Tipuri de adresare : **destinaţie / sursă**
implicit / implicit

Memento de cod : LD SP,rr

cod : F9H

Flaguri afectate : nici unul



Exemple de utilizare :

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 5 : Instrucţiuni de salvare pe stivă a regiştrilor dubli (write memory)

Mnemonica generică : **PUSH rr**

unde : rr poate fi BC, DE, HL, AF, IX sau IY

Total instrucţiuni : 6 * 1 = 6

Acţiunea :

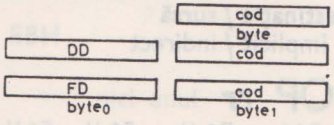
SP = SP − 1
(SP) ⟵ rrhigh
SP = SP − 1
(SP) ⟵ rrlow

Conţinutul registrului dublu **rr** este salvat în memorie la adresa specificată de indicatorul de stivă **SP**. Salvarea se face la adrese descrescătoare, prima salvare implicînd octetul superior al registrului **rr**.
Conţinutul registrului dublu **rr** rămîne nemodificat.
Conţinutul indicatorului de stivă se decrementează cu 2.

Necesar de memorie : 1 byte pentru BC, DE, HL sau AF
2 byte pentru IX sau IY

Necesar de timp : 11 (5,3,3) tacţi procesor pentru BC, DE, HL sau AF
15 (4,5,3,3) tacţi procesor pentru IX sau IY.

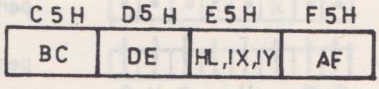
Structura instrucţiunii :



pentru BC, DE, HL sau AF
pentru IX
pentru IY

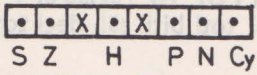
Tipuri de adresare : **destinaţie / sursă**
indirect / implicit

Memento de cod : **PUSH rr**



Flaguri afectate

nici unul



Exemple de utilizare : vol. II, p. 170

Clasa 2 : Instrucţiuni de transfer de 16 bit
Grupa 6 : Instrucţiuni de restaurare a regiştrilor dubli din stivă (read memory)

Mnemonica generică : **POP rr**

unde : rr poate fi BC, DE, HL, AF, IX sau IY

Total instrucţiuni : 6 * 1 = 6

Acţiunea :

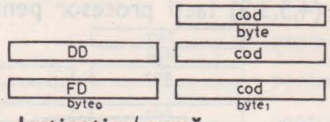
rrlow ⟵ (SP)
SP = SP + 1
rrhigh ⟵ (SP)
SP = SP + 1

Registrul dublu **rr** este încărcat din memorie de la adresa specificată prin indicatorul de stivă **SP**. Conţinutul celulei cu adresă inferioară este transferat în octetul inferior al lui **rr**. Indicatorul de stivă este incrementat cu 2.

Necesar de memorie : 1 byte pentru BC, D,E, HL sau AF
2 byte pentru IX sau IY

Necesar de timp : 10 (4,3,3) tacţi procesor pentru BC, DE, HL sau AF
14 (4,4,3,3) tacţi procesor pentru IX sau IY

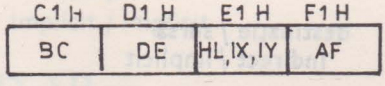
Structura instrucţiunii :



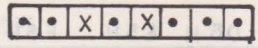
pentru BC, DE, HL sau AF
pentru IX
pentru IY

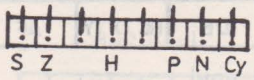
Tipuri de adresare : **destinaţie / sursă**
implicit / indirect

Memento de cod : **POP rr**



Flaguri afectate :

nici unul :  pentru **BC, DE, HL, IX sau IY**

toate :  pentru **AF**

La execuţia instrucţiunii **POP AF** registrul **F** este înscris din memorie de la adresa **(SP)**. Astfel toţi biţii săi, deci toţi indicatorii de condiţii, sînt afectaţi.

Exemple de utilizare : **vol. II, p. 149**

Clasa 2 : Instrucţiuni de transfer de 16 bit

Grupa 7 : Schimb de conţinut între regiştri dubli

Mnemonica : **EX DE,HL**

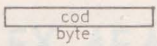
Total instrucţiuni : 1

Acţiunea : D ⟷ H
E ⟷ L

Conţinutul regiştrilor dubli DE şi HL este interschimbat

Necesar de memorie : 1 byte

Necesar de timp : 4(4) tacţi procesor

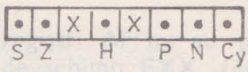
Structura instrucţiunii : 

Tipuri de adresare : **destinaţie / sursă**
implicit / implicit

Memento de cod : EX DE,HL

cod : **EBH**

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p 152, p. 199**

Clasa 2 : Instrucțiuni de transfer de 16 bit

Grupa 8 : Interschimbarea registrilor primari cu cei secundari

Mnemonica : **EXX**

Total instrucțiuni : 1

Acțiunea] : $B \longleftrightarrow B'$
$C \longleftrightarrow C'$
$D \longleftrightarrow D'$
$E \longleftrightarrow E'$
$H \longleftrightarrow H'$
$L \longleftrightarrow L'$

Registri secundari devin registri primari (de lucru) și invers.

Necesar de memorie : 1 byte

Necesar de timp : 4(4) tacți procesor

Structura instrucțiunii : cod / byte

Tipuri de adresare : destinație / sursă
implicit / implicit

Memento de cod : EXX

cod : D9H

Flaguri afectate : nici unul

S	Z		H		P	N	Cy
•	•	X	•	X	•	•	•

Exemple de utilizare : vol. II, p. 147

Clasa 2 : Instrucțiuni de transfer de 16 bit

Grupa 9 : Interschimbarea regiștrilor de stare primari și secundari

Mnemonica : **EX AF,AF'**

Total instrucțiuni : 1

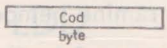
Acțiunea : $A \longleftrightarrow A'$
$F \longleftrightarrow F'$

Conținutul regiștrilor de stare (AF) primari și secundari este interschimbat.

Necesar de memorie : 1 byte

Necesar de timp : 4(4) tacti procesor

Structura instrucțiunii :

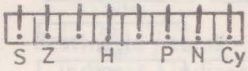


Tipuri de adresare : destinație / sursă
implicit / implicit

Memento de cod : EX AF,AF'

cod : 08H

Flaguri afectate : toate



La execuția instrucțiunii de interschimbare, toți biții registrului F sînt încărcați din registrul secundar F', deci toți indicatorii de condiție sînt afectați.

Observație : regiștri AF nu au putut fi cuprinși în instrucțiunea de schimb EXX, fiindcă astfel execuția instrucțiunii EXX ar fi provocat o ruptură totală, generînd două stări total distincte ale procesorului. Stabilirea unei legături între cele două stări ar fi necesitat transferul unor date (registri) prin memorie, scăzînd astfel eficiența instrucțiunii EXX. Neimplicarea regiștrilor AF în instrucțiunea EXX permite ca ei să fie folosiți ca și „cutie poștală" între cele două stări ale procesorului : pre EXX și post EXX.

Exemple de utilizare : vol. II, p. 150

Clasa 2 : Instrucțiuni de transfer de 16 bit

Grupa 10 : Instrucțiuni de interschimbare a regiștrilor dubli cu vîrful stivei

Mnemonica generică : **EX (SP),rr**

unde : rr poate fi HL, IX sau IY

Total instrucțiuni : $3 * 1 = 3$

Acțiunea : $rr_{low} \longleftrightarrow (SP)$

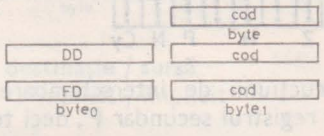
$rr_{high} \longleftrightarrow (SP + 1)$

Conținutul registrului dublu rr este interschimbat cu conținutul a două celule de memorie, adresate prin indicatorul de stivă. Conținutul indicatorului de stivă SP rămîne nemodificat.

Necesar de memorie : 1 byte pentru HL
2 byte pentru IX sau IY

Necesar de timp : 19 (4,3,4,3,5) tacți procesor pentru HL
23 (4,4,3,4,3,5) tacți procesor pentru IX sau IY

Structura instrucțiunii :

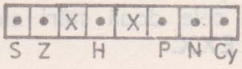


Tipuri de adresare : destinație / sursă
indirect / implicit

Memento de cod : EX (SP),rr

cod : E3H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 199

Clasa 3 : Instrucțiuni de transfer de blocuri de date

Grupa 1 : Transferuri simple

Mnemonica generică : **LD x (LDD sau LDI)**

unde : x poate fi *litera* I sau D

Total instrucțiuni : 1 + 1 = 2

Acțiunea : **LDI** (load & increment)

(DE) ← (HL)
DE = DE + 1
HL = HL + 1
BC = BC − 1

Conținutul celulei de memorie adresată prin registrul dublu HL este transferat în celula de memorie adresată prin registrul dublu DE. Conținutul registrilor DE și HL este incrementat cu 1, iar cel al registrului dublu BC decrementat cu 1.

LDD (load & decrement)

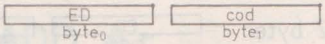
(DE) ← (HL)
DE = DE − 1
HL = HL − 1
BC = BC − 1

Idem, cu deosebirea ca indicatorii de adresă DE și HL sînt decrementați.

Necesar de memorie : 2 byte

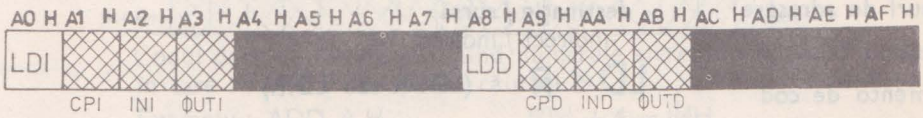
Necesar de timp : 16 (4,4,3,5) tacți procesor

Structura instrucțiunii :

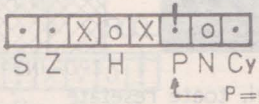


Tipuri de adresare : destinație / sursă
indirect / indirect

Memento de cod : LD (LDD sau LDI)



Flaguri afectate : H, P, N



P=0 dacă după efectuare BC=0
P=1 dacă după efectuare BC≠0

Exemple de utilizare : vol. II, p. 190

Clasa 3 : Instrucţiuni de transfer de blocuri de date

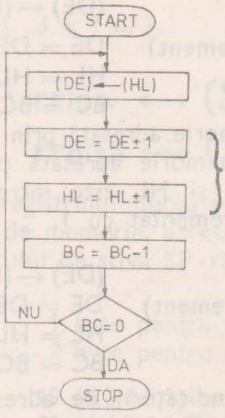
Grupa 2 : Transferuri multiple

Mnemonica generică : **LD × R** (LDDR sau LDIR)

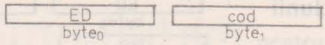
unde : x poate fi *litera* I sau D

Total instrucţiuni : 1 + 1 = 2

Acţiunea :



Instrucţiunea transferă un bloc de date de lungime egală cu **BC** dintr-o zonă de memorie în alta Blocul sursă începe la adresa specificată de registrul **HL.** Blocul destinaţie începe la adresa specificată de **DE.** Transferul are loc în sens *crescător* (LDIR) sau *descrescător* (LDDR).

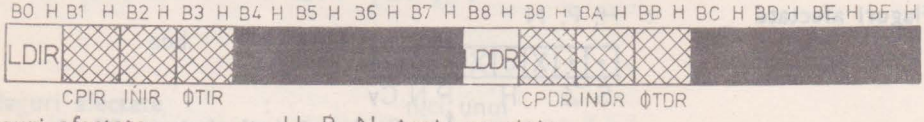
Necesar de memorie : 2 byte 

Necesar de timp : 21 (4,4,3,5,5) taţi procesor dacă BC≠0
16 (4,4,3,5) taţi procesor dacă BC=0

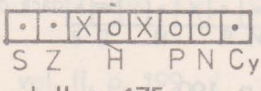
Structura instrucţiunii :

Tipuri de adresare : **destinaţie / sursă**
indirect / indirect

Memento de cod : **LD × R** (LDDR sau LDIR)



Flaguri afectate : H, P, N, toate resetate



Exemple de utilizare : vol. II, p. 175

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit
Grupa 1 : Aritmetice — Adunarea simplă a regiștrilor

Mnemonica generică : **ADD A,r**

unde : r poate fi B, C, D, E, H, L sau A

Total instrucțiuni : $7 * 1 = 7$

Acțiunea : **A = A + r**

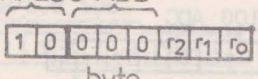
Conținutul registrului r este adunat cu cel al registrului A, rezultatul obținîndu-se în registrul A. Conținutul registrului r nu se modifică.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

Se execută cu suprapunere de cicluri mașină. Fizic necesită 5 tacți.

Structura instrucțiunii

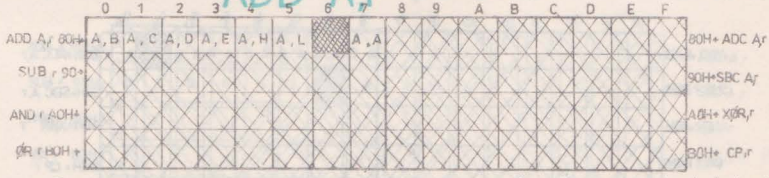


unde : r2r1r0 reprezintă codul pe 3 bit al regiștrilor interni

000 — B	100 — H
001 — C	101 — L
010 — D	110
011 — E	111 — A

Tipuri de adresare : op1 / op2
implicit / implicit

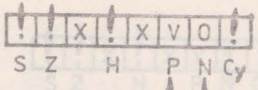
Memento de cod : **ADD A,r**



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : ADD A,H 80H + 4 = 84H

Flaguri afectate : toate



flagul P/V indică depășirea — flagul N=0 indică adunarea efectuată

Exemple de utilizare : vol. II, p. 197

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit
Grupa 2 : Aritmetice — Adunarea regiştrilor cu adăugarea transportului

Mnemonica generică : **ADC A,r** (add with carry)

unde : r poate fi B, C, D, E, H, L sau A

Total instrucţiuni : $7 * 1 = 7$

Acţiunea : $A = A + r + Cy$

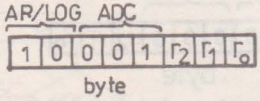
Este o adunare dublă : la conţinutul iniţial al lui A se adaugă conţinutul registrului r şi al flagului carry (transport). Rezultatul se obţine în A. Conţinutul registrului r rămîne neafectat.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Se execută cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :



unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000 — B	100 — H
001 — C	101 — L
010 — D	110
011 — E	111 — A

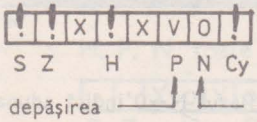
Tipuri de adresare : op1 / op2
implicit / implicit

Memento de cod : **ADC A,r**

Codul dorit se obţine însumînd numerele aferente rîndului şi coloanei în care se află instrucţiunea căutată.

Exemplu : ADC A,D 80H + A = 8AH

Flaguri afectate : toate



Exemple de utilizare :

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit
Grupa 3 : Aritmetice — Scăderea simplă a regiştrilor

Mnemonica generică : **SUB r**

unde : r poate fi B, C, D, E, H, L sau A

Total instrucţiuni : $7 * 1 = 7$

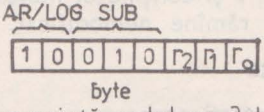
Acţiunea : $A = A - r$

Conţinutul registrului **r** se scade din conţinutul registrului **A.** Rezultatul se obţine în registrul A. Conţinutul registrului r rămîne nemodificat.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor
Se execută cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :

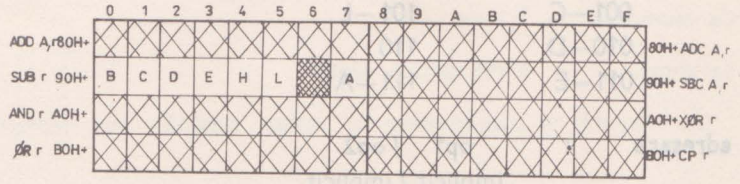


unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000 — B	100 — H
001 — C	101 — L
010 — D	110
011 — E	111 — A

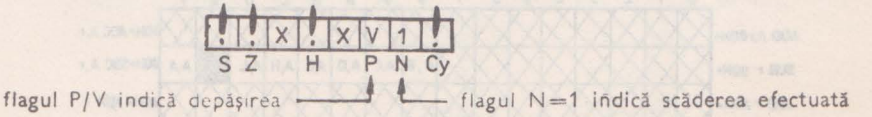
Tipuri de adresare : **op1 / op2**
implicit / implicit

Memento de cod : **SUB r**



Codul dorit se obţine însumînd numerele aferente rîndului şi coloanei în care se află instrucţiunea căutată.
Exemplu : SUB B 90H + 0 = 90H

Flaguri afectate : toate



Exemple de utilizare :

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 4 : Aritmetice — Scăderea regiştrilor cu scăderea transportului

Mnemonica generică : **SBC A,r** (subtract with carry)

unde : r poate fi B, C, D, E, H, L sau A

Total instrucţiuni : 7 * 1 = 7

Acţiunea : $A = A - r - Cy$

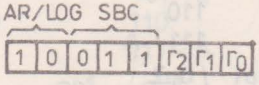
Este o scădere dublă. Din conţinutul iniţial al registrului A se scade conţinutul registrului r şi conţinutul indicatorului de transport. Conţinutul registrului r rămîne nemodificat.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Se execută cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :

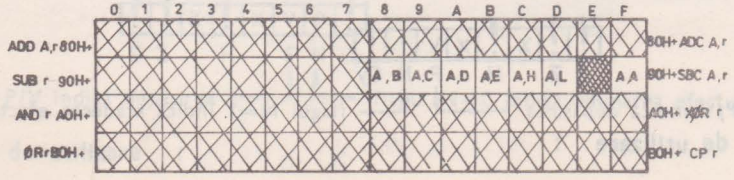


unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : op1 / op2
implicit / implicit

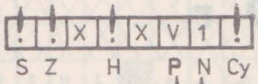
Memento de cod : **SBC A,r**



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : SBC A,E 90H + BH = 9BH

Flaguri afectate : toate



Exemple de utilizare :

Clasa 4 : **Instrucţiuni aritmetice/logice pe 8 bit**

Grupa 5 : **Logice — Operaţia ŞI cu regiştri**

Mnemonica **generică** : **AND r**

unde : r poate fi B, C, D, E, H, L sau A

Total instrucţiuni : 7 * 1 = 7

Acţiunea : $A = A \wedge r$

Se execută o funcţie ŞI logic între conţinutul registrului **A** şi conţinutul registrului r. Operaţia se execută între 2 biţi fiecare avînd aceeaşi poziţie semnificativă. Rezultatul se obţine în registrul **A.** Conţinutul registrului **r** rămîne nemodificat.

		∧
0	0	0
0	1	0
1	0	0
1	1	1

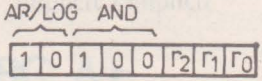
Rezultatul operaţiei ŞI este 1, dacă şi numai dacă ambii biţi au avut valoarea logică 1. În rest rezultatul este 0.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Se execută cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :

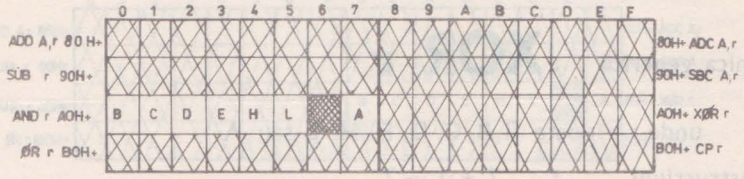


unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : op1 / op2
implicit / implicit

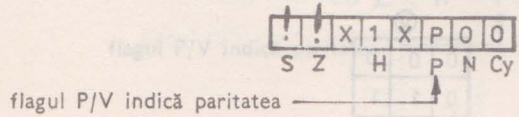
Memento de cod : AND r



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : AND A A0H + 7 = A7H

Flaguri afectate : toate



Exemple de utilizare : **vol. II, p. 174**

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 6 : Logice — Operaţia SAU EXCLUSIV cu regiştri

Mnemonica generică : XOR r

unde : r poate fi B, C, D, E, H, L sau A

Total instrucţiuni : $7 * 1 = 7$

Acţiunea : $A = A \vee r$

Se execută o funcţie **SAU EXCLUSIV** între conţinutul registrului A şi conţinutul registrului r. Operaţia se execută pe cîte 2 biţi, fiecare avînd aceeaşi poziţie semnificativă. Rezultatul se generează în registrul **A.** Conţinutul registrului r rămîne nemodificat.

⊻		
0	0	0
0	1	1
1	0	1
1	1	0

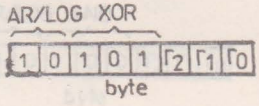
Rezultatul operaţiei SAU EXCLUSIV este 1, dacă şi numai dacă cei doi biţi au avut valori diferite. În caz de egalitate rezultatul este 0.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Se **execută** cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :

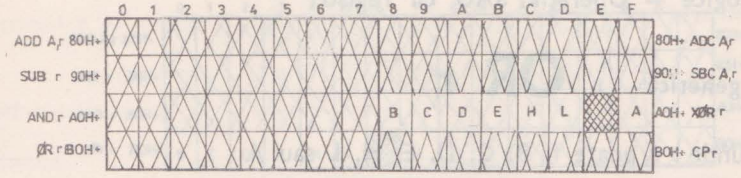


unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : op1 / op2
implicit / implicit

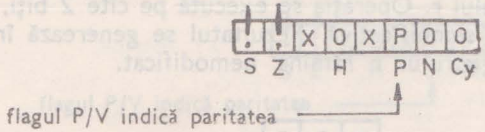
Memento de cod : **XOR r**



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : XOR H A0H + CH = ACH

Flaguri afectate : toate



Exemple de utilizare : vol. II, p. 177

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 7 : Logice — Operaţia SAU cu regiştri

Mnemonica generică : OR r

unde : **r** poate fi **B, C, D, E, H, L** sau **A**

Total instrucţiuni : $7 * 1 = 7$

Acţiunea : $A = A \vee r$

Se execută o funcţie **SAU** logic între conţinutul registrului **A** şi conţinutul registrului **r**. Operaţia se execută pe cîte 2 biţi, fiecare avînd aceeaşi poziţie semnificativă. Rezultatul se generează în registrul **A**. Conţinutul registrului **r** rămîne nemodificat.

		V
0	0	0
0	1	1
1	0	1
1	1	1

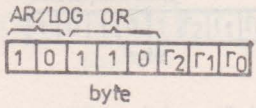
Rezultatul operaţiei SAU este 1, dacă oricare din cei doi biţi are valoarea 1. Rezultatul este 0 dacă ambii biţi au fost 0.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Se execută cu suprapunere de cicluri maşină. Fizic necesită 5 tacţi.

Structura instrucţiunii :

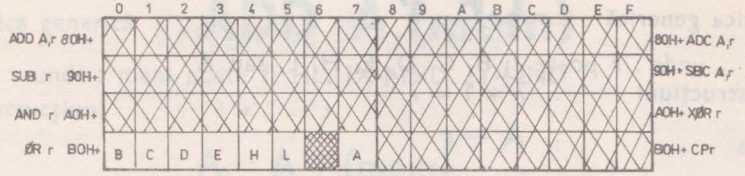


unde : **r2r1r0** reprezintă codul pe 3 bit al **regiştrilor interni**

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : op1 / op2
implicit / implicit

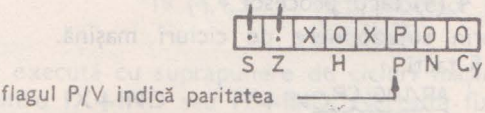
Memento de cod : OR r



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : OR C B0H + 1 = B1H

Flaguri afectate : toate



Exemple de utilizare : vol. II, p. 168, p. 188

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit
Grupa 8 : Logice — Comparaţia acumulatorului cu regiştri

Mnemonica generică : **CP r**

unde : r poate fi B, C, D, E, H, L, sau A

Total instrucţiuni : $7 * 1 = 7$

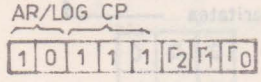
Acţiunea : **A − r**

Se efectuează o scădere. Conţinutul registrului r se scade din conţinutul registrului A.
Operaţia este identică cu cea de la **SUB r** cu deosebirea că nu se generează rezultat. Deci conţinutul regiştrilor A şi r rămîne nemodificat. Se poziţionează indicatorii de condiţie.
Exemplu : dacă A=r atunci flagul zero va fi înscris : Z=1

Necesar de memorie : 1 byte
Necesar de timp : 4 (4) tacţi procesor

Se execută prin suprapunere de cicluri maşină.
Fizic necesită 5 tacţi.

Structura instrucţiunii :

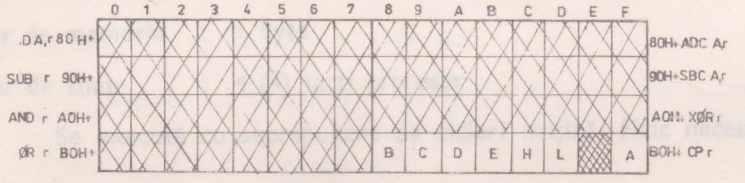


unde : **r2r1r0** reprezintă codul pe 3 bit al regiştrilor interni

000 — B	100 — H	010 — D	110
001 — C	101 — L	011 — E	111 — A

Tipuri de adresare : **op1 / op2**
implicit / implicit

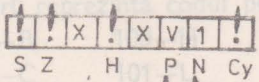
Memento de cod : **CP r**



Codul dorit se obţine însumînd numărul aferent rîndului şi coloanei în care se află instrucţiunea căutată.

Exemplu : CP L B0H + DH = BDH

Flaguri afectate : toate



Exemple de utilizare : vol. II, p. 154

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit
Grupa 9 : Aritmetice — Adunare simplă cu locaţii de memorie

Mnemonica generică : **ADD A,(mem)**

unde : mem poate fi **HL, IX+IND, IY+IND**

Total instrucţiuni : 3 * 1 = 3

Acţiunea : **A = A + (mem)**

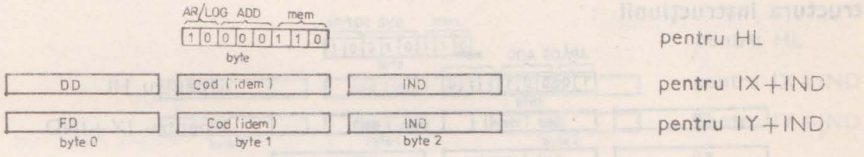
Conţinutul celulei de memorie adresată prin conţinutul registrului dublu este adunat cu conţinutul registrului **A**. Conţinutul celulei de memorie implicată în operaţie nu se schimbă.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacţi procesor pentru HL
19 (4,4,3,5,3) tacţi procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri maşină : M2 pentru **HL** şi M5 pentru **IX+IND** sau **IY+IND**. Necesită fizic 4 tacţi procesor.

Structura instrucţiunii :

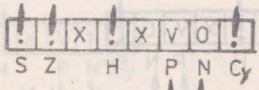


Tipuri de adresare : **op1 / op2**
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : **ADD A,(mem)**

cod : **86H**

Flaguri afectate : toate



flagul P/V indică depăşirea — flagul N=0 indică adunarea efectuată

Exemple de utilizare :

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit
Grupa 10 : Aritmetice — Adunare cu adăugarea transportului (memorie)

Mnemonica generică : **ADC A,(mem)**

unde : **mem** poate fi **HL, IX+IND** sau **IY+IND**

Total instrucțiuni : 3 * 1 = 3

Acțiunea : A = A + (mem) + Cy

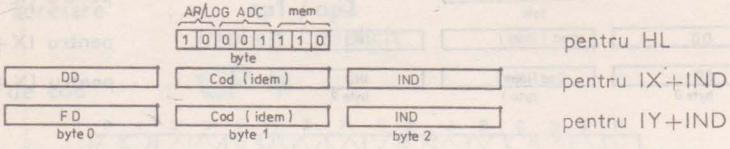
Se execută o adunare dublă : conținutul celulei de memorie, adresată prin conținutul registrului dublu, se adună la conținutul registrului **A** și se adună conținutul indicatorului de transport **(carry)**. Rezultatul se obține în registrul **A**.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacți procesor pentru HL
19 (4,4,3,5,3) tacți procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri mașină : M2 pentru **HL** și M5 pentru **IX+IND** sau **IX+IND** necesită fizic 4 tacți procesor.

Structura instrucțiunii :

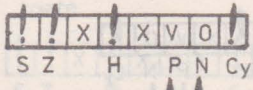


Tipuri de adresare : **op1 / op2**
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : ADC A,(mem)

cod : **8EH**

Flaguri afectate : toate



flagul P/V indică depășirea — flagul N=0 indică adunarea efectuată

Exemple de utilizare : **vol. II, p. 177**

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 11 : Aritmetice — Scădere simplă cu locaţii de memorie

Mnemonica generică : **SUB (mem)**

unde : mem poate fi HL, IX+IND, IY+IND

Total instrucţiuni : 3 * 1 = 3

Acţiunea : A = A − (mem)

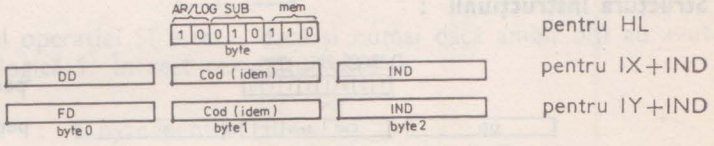
Conţinutul locaţiei de memorie adresată prin registrul dublu este scăzut din conţinutul acumulatorului A. Rezultatul se obţine în registrul A. Conţinutul celulei de memorie implicată în operaţie nu se schimbă.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacţi procesor pentru HL
19 (4,4,3,5,3) tacţi procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri maşină : M2 pentru HL şi M5 pentru IX+IND sau IY+IND necesită fizic 4 tacţi.

Structura instrucţiunii :

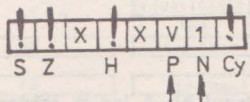


Tipuri de adresare : op1 / op2
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : SUB (mem)

cod : 96H

Flaguri afectate : toate



flagul P/V indică depăşirea — flagul N=1 indică scăderea efectuată

Exemple de utilizare :

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 12 : Aritmetice — Scădere cu scăderea transportului (memorie)

Mnemonica generică : SBC A,(mem)

unde : mem poate fi **HL, IX+IND, IY+IND**

Total instrucţiuni : 3 * 1 = 3

Acţiunea : A = A − (mem) − Cy

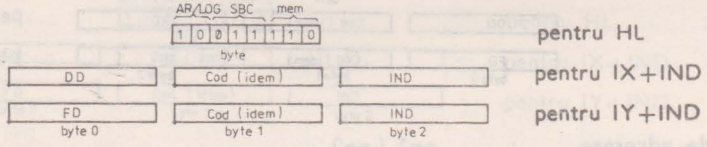
Se efectuează o scădere dublă : conţinutul celulei de memorie adresată prin registrul dublu şi conţinutul indicatorului de transport (carry) se scad din conţinutul registrului A. Rezultatul se obţine în registrul A. Conţinutul celulei de memorie implicată în operaţie nu se schimbă.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacţi procesor pentru HL
19 (4,4,3,5,3) tacţi procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri maşină : M2 pentru **HL** şi M5 pentru **IX+IND** sau **IY+IND** necesită fizic 4 tacţi procesor.

Structura instrucţiunii :

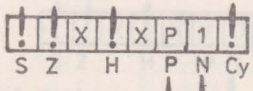


Tipuri de adresare : **op1 / op2**
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : SBC A,(mem)

cod : 9EH

Flaguri afectate : toate



flagul P/V indică depăşirea — flagul N=1 indică scăderea efectuată

Exemple de utilizare : vol. II, p. 178

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 13 : Logice — Operația SI cu locații de memorie

Mnemonica generică : **AND (mem)**

unde : mem poate fi HL, IX+IND sau IY+IND

Total instrucțiuni : 3 * 1 = 3

Acțiunea : **A = A ∧ (mem)**

Se efectuează o funcție SI logic între conținutul registrului A și conținutul celulei de memorie adresată prin registrul dublu. Operația se execută pe cîte 2 biți, fiecare avînd aceeași poziție semnificativă. Rezultatul se obține în registrul A. Conținutul celulei de memorie implicate în operație nu se schimbă.

∧

0	0	0
0	1	0
1	0	0
1	1	1

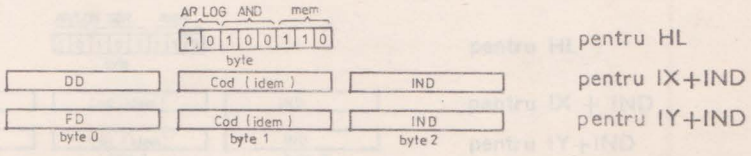
Rezultatul operației SI este 1, dacă și numai dacă ambii biți au avut valoarea logică 1. În rest rezultatul este 0.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacți procesor pentru HL
19 (4,4,3,5,3) tacți procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri mașină : M2 pentru **HL** și M5 pentru **IX+IND** sau IY+IND necesită fizic 4 tacți procesor.

Structura instrucțiunii :

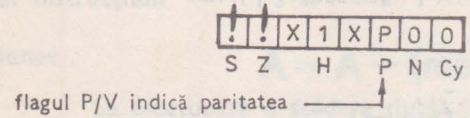


Tipuri de adresare : op1 / op2
implicit / indirect pentru HL
implicit / indexat pentru IX+IND, sau IY+IND

Memento de cod : AND (mem)

cod : A6H

Flaguri afectate : toate



Exemple de utilizare :

Clasa 4 : Instrucţiuni aritmetice /logice pe 8 bit

Grupa 14 : Logice — Operaţia SAU EXCLUSIV cu locaţii de memorie

Mnemonica generică : XOR (mem)

unde : **mem** poate fi **HL, IX+IND, IY+IND**

Total instrucţiuni : $3 * 1 = 3$

Acţiunea : $A = A \vee (mem)$

Se efectuează o funcţie **SAU EXCLUSIV** între conţinutul registrului **A** şi conţinutul celulei de memorie adresată prin registrul dublu. Operaţia se execută pe cîte 2 biţi, fiecare avînd aceeaşi poziţie semnificativă.
Rezultatul se obţine în registrul **A**. Conţinutul celulei de memorie implicată în operaţie nu se schimbă.

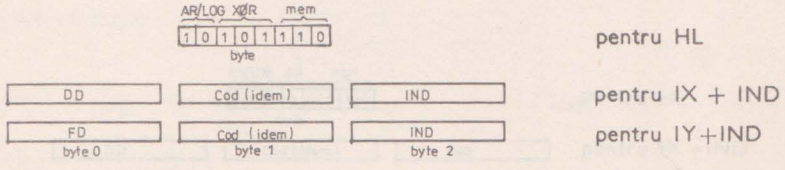
Ⓥ		
0	0	0
0	1	1
1	0	1
1	1	0

Rezultatul operaţiei SAU EXCLUSIV este 1 dacă şi numai dacă cei 2 biţi au avut valori diferite. În caz de egalitate rezultatul este 0.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacţi procesor pentru HL
19 (4,4,3,5,3) tacţi procesor pentru IX+IND sau IY+IND
Se execută cu suprapunere de cicluri maşină. M2 pentru **HL** şi M5 pentru **IX+IND** sau **IY+IND** necesită fizic 4 tacţi procesor.

Structura instrucţiunii :

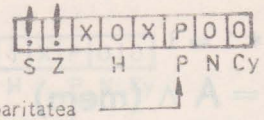


Tipuri de adresare : op1 / op2
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : **XOR (mem)**

cod : **AEH**

Flaguri afectate : toate



flagul P/V indică paritatea

Exemple de utilizare :

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 15 : Logice — Operația SAU cu locații de memorie

Mnemonica generică : **OR (mem)**

unde : mem poate fi HL, IX+IND, IY+IND

Total instrucțiuni : $3 * 1 = 3$

Acțiunea : $A = A \vee (mem)$

Se execută o funcție SAU logic între conținutul registrului A și conținutul celulei de memorie adresată prin registrul dublu. Operația se execută pe cîte 2 biți, fiecare avînd aceeași poziție semnificativă. Rezultatul se generează în A. Conținutul celulei de memorie implicată în operație nu se schimbă.

		V
0	0	0
0	1	1
1	0	1
1	1	1

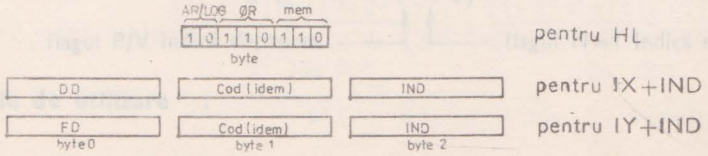
Rezultatul operației SAU este 1, dacă oricare din cei 2 biți are valoarea 1. Rezultatul este 0 doar dacă ambii au fost 0.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacți procesor pentru HL
19 (4,4,3,5,3) tacți procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri mașină : M2 pentru **HL** și M5 pentru **IX+IND** sau **IY+IND** necesită fizic 4 tacți procesor.

Structura instrucțiunii :

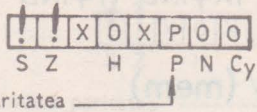


Tipuri de adresare : op1 / op2
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : OR (mem)

cod : B6H

Flaguri afectate : toate



Exemple de utilizare :

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit
Grupa 16 : Logice — Comparaţia acumulatorului cu locaţii de memorie

Mnemonica generică : **CP (mem)**

unde : **mem** poate fi HL, IX+IND, IY+IND

Total instrucţiuni : 3 * 1 = 3

Acţiunea : **A − (mem)**

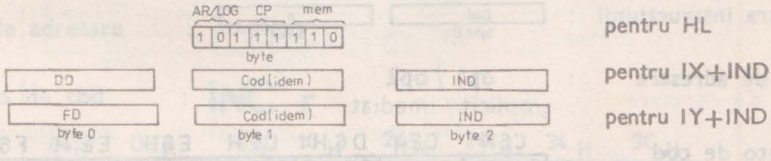
Se efectuează o scădere. Conţinutul celulei de memorie adresată prin conţinutul registrului dublu se scade din conţinutul registrului **A**. Operaţia este similară cu cea de la **SUB (mem)**, cu deosebirea că nu se generează rezultat. Deci conţinutul registrului **A** şi al celulei de memorie implicată în operaţie nu se schimbă. Se poziţionează indicatorii de condiţie.
Exemplu : dacă A < (mem) atunci flagul carry va fi înscris : Cy=1.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 7 (4,3) tacţi procesor pentru HL
19 (4,4,3,5,3) tacţi procesor pentru IX+IND sau IY+IND

Se execută cu suprapunere de cicluri maşină : M2 pentru **HL** şi M5 pentru **IX+IND** sau **IY+IND**. Necesită fizic 4 tacţi procesor.

Structura instrucţiunii :

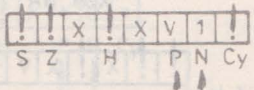


Tipuri de adresare : **op1 / op2**
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : **CP (mem)**

cod : **BEH**

Flaguri afectate :



flagul P/V indică depăşirea — flagul N=1 indică scăderea efectuată

Exemple de utilizare :

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 17 : Instrucțiuni aritmetice și logice imediate

Mnemonice posibile :

ADD A,n	AND n
ADC A,n	XOR n
SUB n	OR n
SBC A,n	CP n

unde : n este un număr [0,255]

Total instrucțiuni : $8 * 1 = 8$

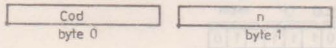
Acțiunea : A = A op n

unde : — op poate fi ADD, ADC, SUB, SBC, AND, XOR sau OR
Oricare din cele șapte operații enunțate se efectuează între conținutul registrului A și numărul n.
Rezultatul se generează în registrul A.
— op CP : A — n ; în cazul operației de comparare.

Necesar de memorie : 2 byte

Necesar de timp : 7 (4,3) tacți procesor
Se execută cu suprapunere de cicluri mașină : M2, necesită fizic 4 tacți procesor.

Structura instrucțiunii :

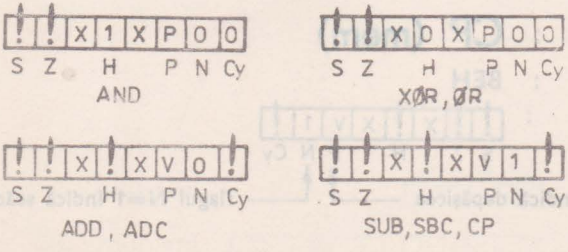


Tipuri de adresare : op1 / op2
implicit / imediat

Memento de cod

C6H	CEH	D6H	DEH	E6H	EEH	F6H	FEH
ADD	ADC	SUB	SBC	AND	XØR	ØR	CP

Flaguri afectate : toate



Exemple de utilizare : vol. II, p. 155, p. 174

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 18 : Incrementarea regiștrilor simpli

Mnemonica generică : **INC r**

unde : r poate fi B, C, D, E, H, L sau A

Total instrucțiuni : 7 * 1 = 7

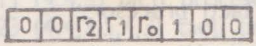
Acțiunea : **r = r + 1**

Conținutul registrului r este incrementat cu 1.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

Structura instrucțiunii :



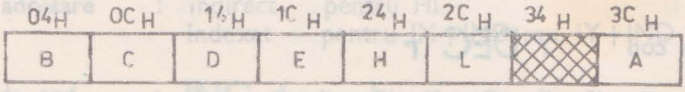
byte

unde : r2r1r0 este codul pe 3 bit al regiștrilor interni

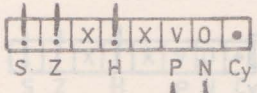
000 — B	100 — H
001 — C	101 — L
010 — D	110
011 — E	111 — A

Tipuri de adresare : implicit

Memento de cod : **INC r**



Flaguri afectate : toate cu excepția lui carry



flagul P/V indică depășirea — flagul N = 0 indică adunarea efectuată

Exemple de utilizare : vol. II, p. 179

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 19 : Decrementarea regiștrilor simpli

Mnemonica generică : DEC r

unde : r poate fi B, C, D, E, H, L sau A

Total instrucțiuni : 7 * 1 = 7

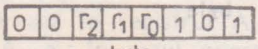
Acțiunea : r = r − 1

Conținutul registrului r este decrementat cu 1.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

Structura instrucțiunii :



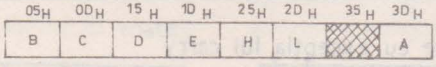
byte

unde : r2r1r0 este codul pe 3 bit al regiștrilor interni

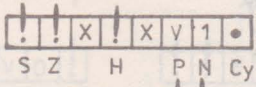
000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : implicit

Memento de cod : DEC r



Flaguri afectate : toate cu excepția lui carry



flagul P/V indică depășirea — flagul N=1 indică scăderea efectuată

Exemple de utilizare : vol. II, p 179

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 20 : Incrementarea locaţiilor de memorie

Mnemonica generică : INC (mem)

unde : mem poate fi HL, IX+IND, IY+IND

Total instrucţiuni : $3 * 1 = 3$

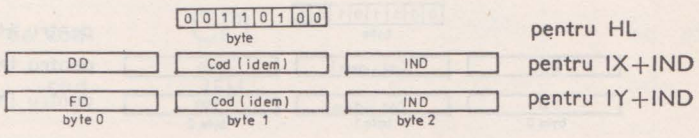
Acţiunea : **(mem) = (mem) + 1**.

Conţinutul celulei de memorie avînd adresa specificată prin conţinutul registrului dublu este incrementat cu 1.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 11 (4,4,3) tacţi procesor pentru HL
23 (4,4,3,5,4,3) tacţi procesor pentru IX+IND sau IY+IND
Ciclul M4 este un ciclu intern pur.

Structura instrucţiunii :

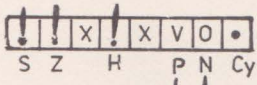


Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : **INC (mem)**

cod : **34H**

Flaguri afectate : toate cu excepţia lui carry



Exemple de utilizare : vol. II, p 182

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 21 : Decrementarea locațiilor de memorie

Mnemonica generică : **DEC (mem)**

unde : mem poate fi HL, IX+IND, IY+IND

Total instrucțiuni : 3 * 1 = 3

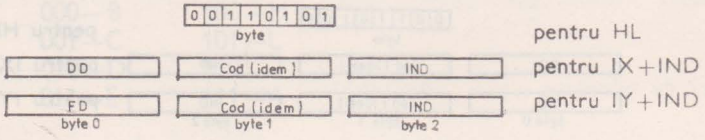
Acțiunea : (mem) = (mem) — 1

Conținutul celulei de memorie adresată prin conținutul registrului dublu este decrementat cu 1.

Necesar de memorie : 1 byte pentru HL
3 byte pentru IX+IND sau IY+IND

Necesar de timp : 11 (4,4,3) tacți procesor pentru HL
23 (4,4,3,5,4,3) tacți procesor pentru IX+IND sau IY+IND. Ciclul M4 este un ciclu intern pur.

Structura instrucțiunii :

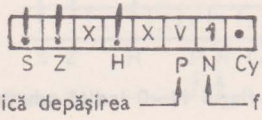


Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : DEC (mem)

cod : 35H

Flaguri afectate : toate cu excepția lui carry



Exemple de utilizare :

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 22 : Complementarea față de 1 (acumulator)

Mnemonica : **CPL**

Total instrucțiuni : 1

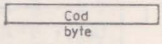
Acțiunea : $A = \bar{A}$

Conținutul acumulatorului A este complementat față de 1. Rezultatul se obține în registrul A.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

Structura instrucțiunii :



Tipuri de adresare : implicit

Memento de cod : CPL

cod : 2FH

Flaguri afectate : H și N devin 1

•	•	X	1	X	•	1	•
S	Z		H		P	N	Cy

Exemple de utilizare : vol. II, p. 145

Clasa 4 : Instrucţiuni aritmetice/logice pe 8 bit

Grupa 23 : Complementarea faţă de 2 (acumulator)

Mnemonica : **NEG**

Total instrucţiuni : 1

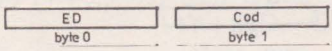
Acţiunea : $A = 0 - A$

Conţinutul acumulatorului A este complementat faţă de 2. Rezultatul se obţine în acumulatorul A.

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacţi procesor

Structura instrucţiunii :

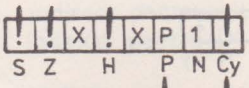


Tipuri de adresare : implicit

Memento de cod : **NEG**

cod : **44H**

Flaguri afectate : toate



P=1 dacă A a fost 80H

Cy=1 dacă A a fost 0

Exemple de utilizare :

Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit
Grupa 24 : Ajustarea BCD a acumulatorului

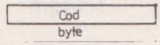
Mnemonica : **DAA**

Total instrucțiuni : 1

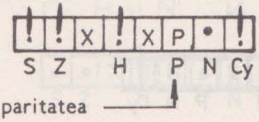
Acțiunea :

Dacă se folosește una din instrucțiunile **ADD, ADC, INC, SUB, SBC, DEC** sau **NEG** instrucțiunea **DAA** ajustează sub forma BCD conținutul registrului **A**. Se presupune că numerele implicate în operația premergătoare au fost deja numere BCD. Pentru a realiza funcția dorită se adaugă +6, +60, +66, —6, —60 sau —66 la conținutul acumulatorului în funcție de starea flagurilor C, N și H. Această instrucțiune impune existența flagurilor N și H.
Redăm tabela de operații publicată în [13, p. 236]

N	Cy	digitul superior	H	digitul inferior	Se adună	C după DAA
0	0	0-9	0	0-9	00	0
după	0	0-8	0	A-F	06	0
ADD	0	0-9	1	0-3	06	0
ADC	0	A-F	0	0-9	60	1
INC	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
1	0	0-9	0	0-9	00	0
după SUB	0	0-8	1	6-F	FA (-6)	0
SBC, DEC	1	7-F	0	0-9	A0 (-60)	1
NEG	1	6-F	1	6-F	9A (-66)	1

Necesar de memorie : 1 byte
Necesar de timp : 4 (4) tacți procesor
Structura instrucțiunii : 

Tipuri de adresare : implicit
Memento de cod : **DAA**
cod : **27H**
Flaguri afectate : toate cu excepția lui N



Exemple de utilizare : vol. II, p. 177

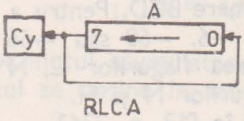
Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

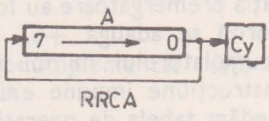
Grupa 25 : Rotirea acumulatorului „cu" carry

Mnemonica generică : **R x CA** (RLCA sau RRCA)

unde : x poate fi *litera* L sau R

Total instrucțiuni : 1 + 1 = 2





Acțiunea : RLCA

Conținutul registrului A este deplasat cu o poziție la **stînga.**
Bitul 7 inițial se mută atît în Cy cît și pe poziția bit 0.

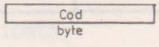
RRCA

Conținutul registrului A este deplasat cu o poziție la **dreapta.**
Bitul 0 inițial se mută atît în Cy cît și pe poziția bit 7.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor
Se execută cu suprapunere de cicluri mașină. Fizic necesită 5 tacți.

Structura instrucțiunii :

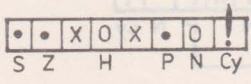


Tipuri de adresare : implicit

Memento de cod : R x CA (RLCA sau RRCA)

	RLCA	RRCA
cod :	07H	0FH

Flaguri afectate : Cy, H și N



Exemple de utilizare : vol. II, p. 186

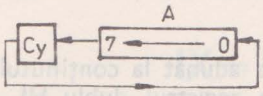
Clasa 4 : Instrucțiuni aritmetice/logice pe 8 bit

Grupa 26 : Rotirea acumulatorului „prin" carry

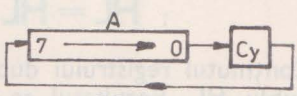
Mnemonica generică : **R × A** (RLA sau RRA)

unde : x poate fi litera L sau R

Total instrucțiuni : 1 + 1 = 2



RLA



RRA

Acțiunea : **RLA**

Conținutul acumulatorului **A** este rotit cu o poziție la **stînga.**
Bitul **7** se mută în **Cy**. **Cy** se mută în bit **0.**

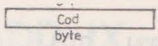
RRA

Conținutul acumulatorului **A** este rotit cu o poziție la **dreapta.**
Bitul **0** se mută în **Cy**. **Cy** se mută în bit **7.**

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor
Se execută cu suprapunere de cicluri mașină. Fizic necesită 5 tacți.

Structura instrucțiunii :

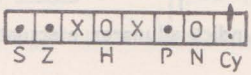


Tipuri de adresare : implicit

Memento de cod : **R × A** (RLA sau RRA)

	RLA	RRA
cod	17H	1FH

Flaguri afectate : Cy, H și N



Exemple de utilizare : **vol. II, p. 202**

Clasa 5 : Instrucţiuni aritmetice pe 16 bit

Grupa 1 : Adunare simplă cu acumulatorul în HL

Mnemonica generică : **ADD HL,rr**

unde : **rr** poate fi **BC, DE, HL** sau **SP**

Total instrucţiuni : 4 * 1 = 4

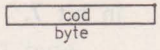
Acţiunea : **HL = HL + rr**

Conţinutul registrului dublu **rr** este adunat la conţinutul registrului dublu **HL**. Rezultatul se obţine în registrul dublu **HL**. Conţinutul registrului dublu **rr** rămîne neschimbat.

Necesar de memorie : 1 byte

Necesar de timp : 11 (4,4,3) tacţi procesor

Structura instrucţiunii :

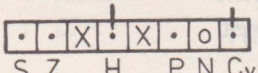


Tipuri de adresare : **op1 / op2**
implicit / implicit

Memento de cod : **ADD HL,rr**

09 H	19 H	29 H	39 H
HL,BC	HL,DE	HL,HL	HL,SP

Flaguri afectate : Cy, N, H



flagul H=1, dacă a existat transport de la bit 11

flagul Cy=1, dacă a existat transport de la bit 15

flagul N=0 indică adunarea efectuată

Exemple de utilizare : **vol. II, p. 170**

Clasa 5 : Instrucțiuni aritmetice pe 16 bit

Grupa 2 : Adunare simplă cu acumulatorul în IX

Mnemonica generică : **ADD IX,rr**

unde : **rr** poate fi **BC, DE, IX** sau **SP**

Total instrucțiuni : 4 * 1 = 4

Acțiunea : **IX = IX + rr**

Conținutul registrului dublu **rr** este adunat la conținutul registrului dublu **IX**. Rezultatul se obține în registrul dublu **IX**. Conținutul registrului dublu **rr** rămîne neschimbat.

Necesar de memorie : 2 byte

Necesar de timp : 15 (4,4,4,3) tacți procesor

Structura instrucțiunii :

DD	cod
$byte_0$	$byte_1$

Tipuri de adresare : **op1 / op2**
implicit / implicit

Memento de cod : **ADD IX,rr**

09H	19H	29H	39H
IX,BC	IX,DE	IX,IX	IX,SP

Flaguri afectate : Cy, N, H

Exemple de utilizare : **vol. II, p. 174**

Clasa 5 : Instrucţiuni aritmetice pe 16 bit

Grupa 3 : Adunare simplă cu acumulatorul în IY

Mnemonica generică : **ADD IY,rr**

unde : **rr** poate fi **BC**, **DE**, **IY** sau **SP**

Total instrucţiuni : $4 * 1 = 4$

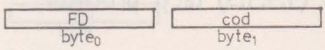
Acţiunea : **IY = IY + rr**

Conţinutul registrului dublu **rr** este adunat la conţinutul registrului dublu **IY**. Rezultatul se obţine în registrul dublu **IY**. Conţinutul registrului dublu **rr** rămîne neschimbat.

Necesar de memorie : 2 byte

Necesar de timp : 15 (4,4,4,3) tacţi procesor

Structura instrucţiunii :

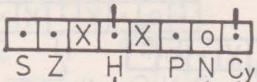


Tipuri de adresare : **op1** / **op2**
implicit / implicit

Memento de cod : **ADD IY,rr**

09 H	19 H	29 H	39 H
IY,BC	IY,DE	IY,IY	IY,SP

Flaguri afectate : Cy, N, H



flagul H=1, dacă a existat tansport de la bit 11

flagul Cy=1, dacă a existat transport de la bit 15

flagul N=0 indică adunarea efectuată

Exemple de utilizare :

Clasa 5 : Instrucţiuni aritmetice pe 16 bit

Grupa 4 : Adunare cu adunarea transportului (carry)

Mnemonica generică : **ADC HL,rr**

unde : **rr** poate fi **BC**, **DE**, **HL** sau **SP**

Total instrucţiuni : $4 * 1 = 4$

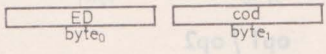
Acţiunea : $HL = HL + rr + Cy$

Se efectuează două adunări : conţinutul registrului dublu **rr** se adună la conţinutul registrului dublu **HL**. La suma astfel obţinută se adună conţinutul indicatorului de transport **(carry)**. Rezultatul se obţine în registrul dublu **HL**. Conţinutul registrului dublu **rr** rămîne neschimbat.

Necesar de memorie : 2 byte

Necesar de timp : 15 (4,4,4,3) tacţi procesor

Structura instrucţiunii :

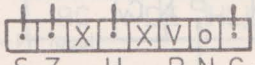


Tipuri de adresare : **op1 / op2**
implicit / implicit

Memento de cod : **ADC HL, rr**

4AH	5AH	6AH	7AH
HL,BC	HL,DE	HL,HL	HL,SP

Flaguri afectate : toate



flagul H=1, dacă a existat transport de la bit 11

flagul N=0 indică adunarea efectuată

flagul P/V indică depăşirea

Exemple de utilizare : **vol. II, p. 202**

Clasa 5 : Instrucțiuni aritmetice pe 16 bit

Grupa 5 : Scădere cu scăderea transportului (carry)

Mnemonica generică : **SBC HL,rr**

unde : rr poate fi BC, DE, HL sau SP

Total instrucțiuni : 4 * 1 = 4

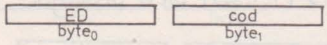
Acțiunea : **HL = HL − rr − Cy**

Se efectuează două scăderi : conținutul registrului dublu **rr** se scade din conținutul registrului dublu **HL**. Din diferența astfel obținută se scade conținutul indicatorului de transport. Rezultatul se obține în registrul dublu **HL**. Conținutul registrului dublu **rr** rămîne neschimbat.

Necesar de memorie : 2 byte

Necesar de timp : 15 (4,4,4,3) tacți procesor

Structura instrucțiunii

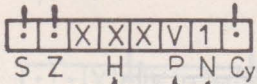


Tipuri de adresare : **op1 / op2**
implicit / implicit

Memento de cod : **SBC HL,rr**

42H	52H	62H	72H
HL,BC	HL,DE	HL,HL	HL,SP

Flaguri afectate : toate



flagul H=1, dacă a existat transport de la bit 11

flagul P/V indică depășirea

flagul N=1 indică scăderea efectuată

Exemple de utilizare : **vol. II, p. 202**

Clasa 5 : Instrucțiuni aritmetice pe 16 bit

Grupa 6 : Incrementarea registrilor dubli

Mnemonica generică : **INC rr**

unde : **rr** poate fi **BC, DE, HL, IX, IY,** sau **SP**

Total instrucțiuni : $6 * 1 = 6$

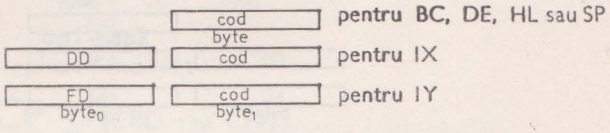
Acțiune : $rr = rr + 1$

Conținutul registrului dublu **rr** este incrementat cu 1.

Necesar de memorie : 1 byte pentru BC, DE, HL sau SP
2 byte pentru IX sau IY

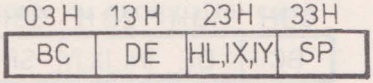
Necesar de timp : 6 (6) tacți procesor pentru BC, DE, HL sau SP
10 (4,6) tacți procesor pentru IX sau IY

Structura instrucțiunii :

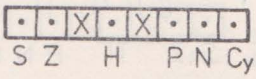


Tipuri de adresare : implicit

Memento de cod : **INC rr**



Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 195

Clasa 5 : Instrucțiuni aritmetice pe 16 bit

Grupa 7 : Decrementarea regiștrilor dubli

Mnemonica generică : **DEC rr**

unde : rr poate fi BC, DE, HL, IX, IY sau SP

Total instrucțiuni : $6 * 1 = 6$

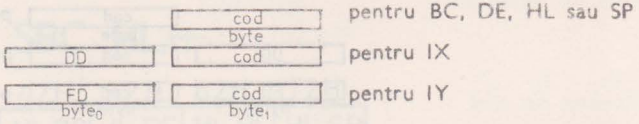
Acțiunea : rr ← rr − 1

Conținutul registrului dublu rr este decrementat cu 1.

Necesar de memorie : 1 byte pentru BC, DE, HL sau SP
2 byte pentru IX sau IY

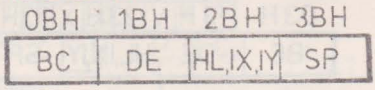
Necesar de timp : 6 (6) tacți procesor pentru BC, DE, HL sau SP
10 (4,6) tacți procesor pentru IX sau IY

Structura instrucțiunii :

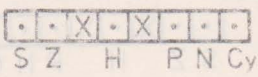


Tipuri de adresare : implicit

Memento de cod : DEC rr



Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 178**

Clasa 6 : Instrucţiuni logice pe blocuri de date

Grupa 1 : Comparări simple

Mnemonica generică : **CP × (CPD sau CPI)**

unde : x poate fi *litera* I sau D

Total instrucţiuni : 1 + 1 = 2

Acţiunea : **CPI** (compare&increment)

A — (HL)
HL = HL + 1
BC = BC — 1

Conţinutul acumulatorului **A** este comparat cu cel al celulei de memorie adresată prin registrul dublu **HL**. Indicatorul de adresă **HL** este incrementat cu 1 iar numărătorul de octeţi **BC** este decrementat cu 1. Conţinutul acumulatorului **A** nu se schimbă. Rezultatul comparaţiei se regăseşte în registrul de flag **F**.

CPD (compare&decrement)

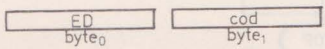
A — (HL)
HL = HL — 1
BC = BC — 1

Idem cu **CPI**, dar indicatorul de adresă **HL** este decrementat cu 1.

Necesar de memorie : 2 byte

Necesar de timp : 16 (4,4,3,5) tacţi procesor

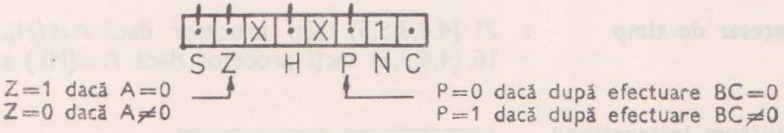
Structura instrucţiunii :



Tipuri de adresare : **op1 / op2**
implicit / indirect

Memento de cod : **CP X (CPD sau CPI)**

Flaguri afectate : toate cu excepţia lui carry



Exemple de utilizare : vol. II, p. 147

Clasa 6 : Instrucţiuni logice pe blocuri de date

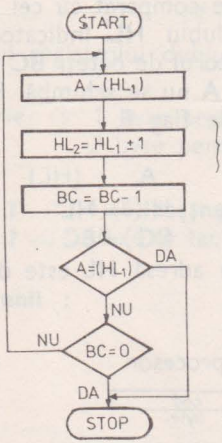
Grupa 2 : Comparări multiple

Mnemonica generică : CP × R (CPDR sau CPIR)

unde : x poate fi *litera* I sau D

Total instrucţiuni : 1 + 1 = 2

Acţiunea

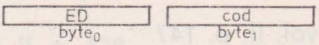


+ CPIR — compare, increment, repeat
— CPDR — compare, decrement, repeat

Conţinutul acumulatorului **A** este comparat cu cel al celulei de memorie specificată prin registrul dublu **HL**. Indicatorul de adresă **HL** este *incrementat* (CPIR) sau *decrementat* (CPDR) cu 1. Numărătorul de octeţi BC este decrementat cu 1. Dacă rezultatul comparării arată egalitate instrucţiunea se termină, dacă nu, ea va fi reluată pînă cînd **BC=0**. Conţinutul acumulatorului **A** rămîne nemodificat.

Necesar de memorie : 2 byte

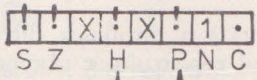
Necesar de timp : 21 (4,4,3,5,5) tacţi procesor dacă A≠(HL) şi BC≠0
16 (4,4,3,5) tacţi procesor dacă A=(HL) sau BC=0

Structura instrucţiunii : 

Tipuri de adresare : op1 / op2
implicit / indirect

Memento de cod : CP × R (CPDR sau CPIR)

Flaguri afectate : toate cu excepția lui carry



Z=1, dacă A=(HL)
Z=0, dacă A≠(HL)

P=0, dacă după efectuare BC=0
P=1, dacă după efectuare BC≠0

Exemple de utilizare :

Clasa 7 : Instrucțiuni orientate pe bit
Grupa 1 : Testarea biților din regiștri

Mnemonica generică : **BIT x,r**

unde : x este numărul bitului [0,7]
r poate fi **B, C, D, E, H, L** sau **A**

Total instrucțiuni : $8 * 7 = 56$

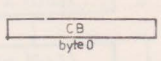
Acțiunea : $Z = \overline{bit\ x_r}$

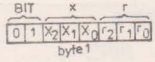
Valoarea complementată a bitului **x** din registrul **r** este copiată în flagul **Z**. Conținutul registrului **r** rămîne neschimbat.

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacți procesor

Structura instrucțiunii :





unde : **x2x1x0** este codul pe 3 bit al bitului testat
r2r1r0 este codul pe 3 bit al registrului testat

000 — B 100 — H
001 — C 101 — L
010 — D 110
011 — E 111 — A

Tipuri de adresare : op1 / op2
implicit / implicit

Memento de cod : **BIT x,r**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
BIT 0,r 40H+	0,B	0,C	0,D	0,E	0,H	0,L		0,A	1,B	1,C	1,D	1,E	1,H	1,L		1,A	40H+BIT 1,r
BIT 2,r 50H+	2,B	2,C	2,D	2,E	2,H	2,L		2,A	3,B	3,C	3,D	3,E	3,H	3,L		3,A	50H+BIT 3,r
BIT 4,r 60H+	4,B	4,C	4,D	4,E	4,H	4,L		4,A	5,B	5,C	5,D	5,E	5,H	5,L		5,A	60H+BIT 5,r
BIT 6,r 70H+	6,B	6,C	6,D	6,E	6,H	6,L		6,A	7,B	7,C	7,D	7,E	7,H	7,L		7,A	70H+BIT 7,r

Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.
Ex : BIT 3,E 50H + BH = 5BH

Flaguri afectate : Z, H, N

•	!	X	1	X	•	0	•
S	Z		H		P	N	C_y

Exemple de utilizare : vol. II, p. 156

Clasa 7 : Instrucţiuni orientate pe bit

Grupa 2 : Inscrierea (set) biţilor din regiştri

Mnemonica generică : **SET x,r**

unde : x este numărul bitului [0,7]
r poate fi B, C, D, E, H, L, sau A

Total instrucţiuni : $8 \times 7 = 56$

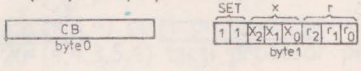
Acţiunea : **bit $x_r = 1$**

În bitul cu numărul x din registrul r se înscrie valoarea 1.

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacţi procesor

Structura instrucţiunii :



unde : x2x1x0 este codul pe 3 bit al bitului setat
r2r1r0 este codul pe 3 bit al regiştrilor interni

000 — B	100 — H
001 — C	101 — L
010 — D	110
011 — E	111 — A

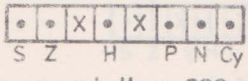
Tipuri de adresare : op1 / op2
implicit / implicit

Memento de cod : **SET x,r**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
SET 0,r COH+	0,B	0,C	0,D	0,E	0,H	0,L		0,A	1,B	1,C	1,D	1,E	1,H	1,L		1,A	COH+SET 1,r
SET 2,r DOH+	2,B	2,C	2,D	2,E	2,H	2,L		2,A	3,B	3,C	3,D	3,E	3,H	3,L		3,A	DOH+SET 3,r
SET 4,r EOH+	4,B	4,C	4,D	4,E	4,H	4,L		4,A	5,B	5,C	5,D	5,E	5,H	5,L		5,A	EOH+SET 5,r
SET 6,r FOH+	6,B	6,C	6,D	6,E	6,H	6,L		6,A	7,B	7,C	7,D	7,E	7,H	7,L		7,A	FOH+SET 7,r

Codul dorit se obţine însumînd numerele aferente rîndului şi coloanei în care se află instrucţiunea căutată.
Ex : SET 6,B FOH + 0 = FOH

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 200

Clasa 7 : Instrucțiuni orientate pe bit

Grupa 3 : Ștergerea (reset) biților din regiștri

Mnemonica generică : **RES x,r**

unde : **x** este numărul bitului [0,7]
r poate fi **B, C, D, E, H, L** sau **A**

Total instrucțiuni : $8 \times 7 = 56$

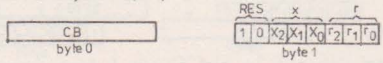
Acțiunea : **bit $x_r = 0$**

În bitul cu numărul **x** din registrul **r** se înscrie valoarea 0.

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacți procesor

Structura instrucțiunii



unde : **x2x1x0** este codul pe 3 bit al bitului resetat
r2r1r0 este codul pe 3 bit al regiștrilor interni.

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

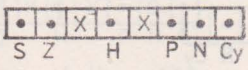
Tipuri de adresare : op1 / op2
implicit / implicit

Memento de cod : **RES x,r**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
RES 0,r	80H+	0,B	0,C	0,D	0,E	0,H	0,L		0,A	1,B	1,C	1,D	1,E	1,H	1,L		1,A	80H+RES 1,r
RES 2,r	90H+	2,B	2,C	2,D	2,E	2,H	2,L		2,A	3,B	3,C	3,D	3,E	3,H	3,L		3,A	90H+RES 3,r
RES 4,r	A0H+	4,B	4,C	4,D	4,E	4,H	4,L		4,A	5,B	5,C	5,D	5,E	5,H	5,L		5,A	A0H+RES 5,r
RES 6,r	B0H+	6,B	6,C	6,D	6,E	6,H	6,L		6,A	7,B	7,C	7,D	7,E	7,H	7,L		7,A	B0H+RES 7,r

Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.
Ex : RES 7,A B0H + FH = BFH

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 156

Clasa 7 : Instrucţiuni orientate pe bit
Grupa 4 : Testarea biţilor din locaţii de memorie

Mnemonica generică : **BIT x,(mem)**

unde : x este numărul bitului [0,7]
mem poate fi **HL, IX .IND** sau **IY+IND**

Total instrucţiuni : $8 * 3 = 24$

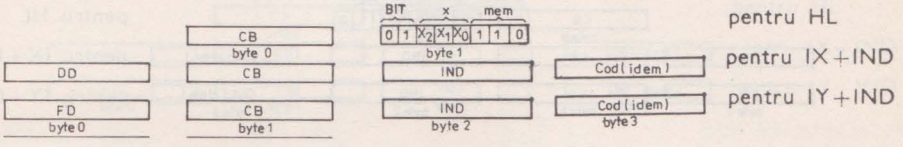
Acţiunea : $Z = \overline{bit\ x_{(mem)}}$

Valoarea complementată a bitului **x** din celula de memorie adresată prin **mem** este copiată în flagul **Z**. Conţinutul celulei de memorie rămîne neschimbat.

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 12 (4,4,4) tacţi procesor pentru HL
20 (4,4,3,5,4) tacţi procesor pentru IX+IND sau IY+IND

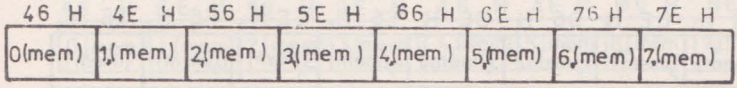
Structura instrucţiunii :



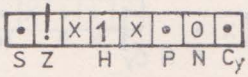
unde : **x2x1x0** este codul pe 3 bit al bitului testat

Tipuri de adresare : **op1 / op2**
implicit / indirect pentru HL
implicit / indexat pentru IX+IND sau IY+IND

Memento de cod : **BIT x,(mem)**



Flaguri afectate : H, N



Exemple de utilizare :

Clasa 7 : Instrucţiuni orientate pe bit
Grupa 5 : Înscrierea (set) biţilor din locaţii de memorie

Mnemonica generică : **SET x,(mem)**

unde : **x** este numărul bitului [0,7]
mem poate fi **HL**, **IX+IND** sau **IY+IND**

Total instrucţiuni : 8 * 3 = 24

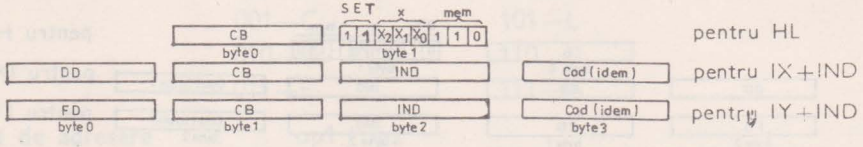
Acţiunea : **bit $x_{(mem)}$ = 1**

În bitul cu numărul **x** din celula de memorie adresată prin **mem**, se înscrie valoarea 1.

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacţi procesor pentru HL
23 (4,4,3,5,4,3) tacţi procesor pentru IX+IND sau IY+IND

Structura instrucţiunii :



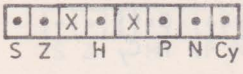
unde : **x2x1x0** este codul pe 3 bit al bitului setat

Tipuri de adresare : **op1 / op2**
implicit / indirect — pentru HL
implicit / indexat — pentru IX+IND sau IY+IND

Memento de cod : **SET x,(mem)**

C6 H	CE H	D6 H	DE H	E6 H	EE H	F6 H	FE H
0,(mem)	1,(mem)	2,(mem)	3,(mem)	4,(mem)	5,(mem)	6,(mem)	7,(mem)

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 189**

Clasa 7 : Instrucțiuni orientate pe bit

Grupa 6 : Ștergerea (reset) biților din locații de memorie

Mnemonica generică : **RES x,(mem)**

unde : x este numărul bitului [0,7]
mem poate fi HL, IX+IND sau IY+IND

Total instrucțiuni : 8 * 3 = 24

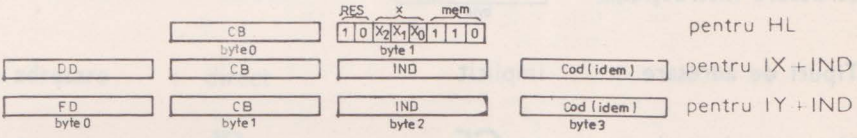
Acțiunea : **bit $x_{(mem)}$ = 0**

În bitul cu numărul x din celula de memorie adresată prin **mem**, se înscrie valoarea 0.

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacți procesor pentru HL
23 (4,4,3,5,4,3) tacți procesor pentru IX+IND sau IY+IND

Structura instrucțiunii :



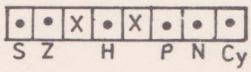
unde : x2x1x0 este codul pe 3 bit al bitului resetat

Tipuri de adresare : op1 / op2
implicit / indirect — pentru HL
implicit / indexat — pentru IX+IND sau IY+IND

Memento de cod : **RES x,(mem)**

86 H	8E H	96 H	9E H	A6 H	AE H	B6 H	BE H
0,(mem)	1,(mem)	2,(mem)	3,(mem)	4,(mem)	5,(mem)	6,(mem)	7,(mem)

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 190

Clasa 7 : Instrucțiuni orientate pe bit

Grupa 7 : Manipularea indicatorului de transport (carry)

Mnemonica generică : × **CF** (SCF sau CCF)

unde : x poate fi *litera* S sau C

Total instrucțiuni : 1 + 1 = 2

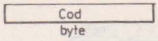
Acțiunea : SCF Cy = 1 înscrie indicatorul de transport

CCF $Cy=\overline{Cy}$ complementează indicatorul de transport

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

Structura instrucțiunii :



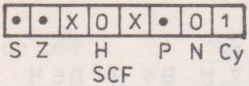
Tipuri de adresare : implicit

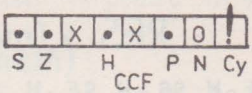
Memento de cod : × CF

SCF 37H

CCF 3FH

Flaguri afectate :





Exemple de utilizare : vol. II, p. 168

Clasa 8 : Instrucţiuni de salt

Grupa 1 : Salt necondiţionat absolut (direct)

Mnemonica : **JP nn**

unde : **nn** este o adresă [0,65535]

Total instrucţiuni : 1

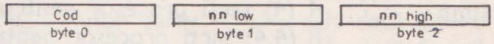
Acţiunea : **PC = nn**

În contorul program **PC**, se înscrie adresa **nn**. Programul se va executa în continuare de la adresa **nn**.

Necesar de memorie : 3 byte

Necesar de timp : 10 (4,3,3) tacţi procesor

Structura instrucţiunii :

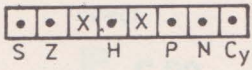


Tipuri de adresare : direct

Memento de cod : **JP nn**

cod : **C3H**

Flaguri afectate : nici unul



Exemple de utilizare :

Clasa 8 : Instrucţiuni de salt

Grupa 2 : Salt necondiţionat absolut (indirect)

Mnemonica generică : **JP (rr)**

unde: **rr** poate fi **HL, IX** sau **IY**

Total instrucţiuni : 3 * 1 = 3

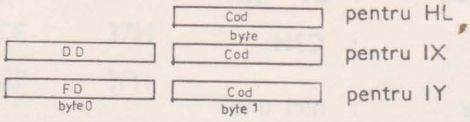
Acţiunea : **PC = rr**

În contorul program **PC**, se copiază conţinutul registrului dublu **rr**. Programul se va executa în continuare de la adresa specificată prin **rr**. Conţinutul registrului dublu **rr** nu se modifică.

Necesar de memorie : 1 byte pentru HL
2 byte pentru IX sau IY

Necesar de timp : 4 (4) tacţi procesor pentru HL
8 (4,4) tacţi procesor pentru IX sau IY

Structura instrucţiunii :

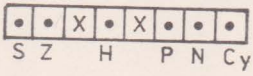


Tipuri de adresare : indirect

Memento de cod : **JP (rr)**

cod : **E9H**

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 152**

Clasa 8 : Instrucţiuni de salt

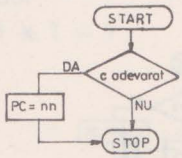
Grupa 3 : Salt condiţionat absolut (direct)

Mnemonica generică : **JP c,nn**

unde : c este condiţia şi poate fi NZ, Z, NC, C, PO, PE, P sau M

Total instrucţiuni : 8 * 1 = 8

Acţiunea :



Se testează cîte un bit al registrului F. Dacă condiţia căutată este adevărată se efectuează saltul (PC=nn). În caz contrar programul continuă cu instrucţiunea următoare.

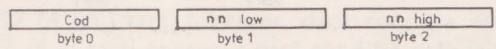
Semnificaţia condiţiilor :

NZ	— non zero	→ salt dacă Z=0
Z	— zero	→ salt dacă Z=1
NC	— non carry	→ salt dacă Cy=0
C	— carry	→ salt dacă Cy=1
PO	— parity odd	→ salt dacă P/V=0
PE	— parity even	→ salt dacă P/V=1
P	— plus	→ salt dacă S=0
M	— minus	→ salt dacă S=1

Necesar de memorie : 3 byte

Necesar de timp : 10 (4,3,3) tacţi procesor

Structura instrucţiunii :

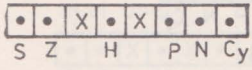


Tipuri de adresare : direct

Memento de cod : **JP c,nn**

	$C2_H$	CA_H	$D2_H$	DA_H	$E2_H$	EA_H	$F2_H$	FA_H
JP c	NZ	Z	NC	C	PO	PE	P	M

Flaguri afectate : nici unul



Exemple de utilizare : **vol. II, p. 147**

Clasa 8 : Instrucțiuni de salt

Grupa 4 : Salt necondiționat relativ

Mnemonica generică : **JR d**

unde : **d** este deplasamentul [—128, +127] exprimat în complementul față de doi

Total instrucțiuni : 1

Acțiune : **PC = PC + d**

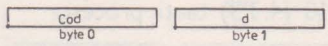
Deplasamentul **d** (număr cu semn) este adunat la valoarea curentă a contorului program **PC**. Se calculează astfel noua adresă, de la care se va continua execuția programului.

Necesar de memorie : 2 byte

Observație : În momentul în care se efectuează calculul adresei de salt, contorul program **PC** conține deja adresa instrucțiunii următoare, adresa care este cu +2 mai mare decît adresa instrucțiunii de salt însăși. De aceea plaja de adrese ce poate fi acoperită prin **JR** este de [—126, +129] față de adresa de început a instrucțiunii de salt.

Necesar de timp : 12 (4,3,5) tacți procesor

Structura instrucțiunii :

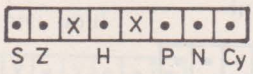


Tipuri de adresare : imediat (relativ)

Memento de cod **JR d**

cod : **18H**

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 155

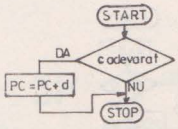
Clasa 8 : Instrucțiuni de salt
Grupa 5 : Salt condiționat relativ

Mnemonica generică : **JR c,d**

unde : c este condiția și poate fi NZ, Z, NC sau C
d este deplasamentul [—128, +127] exprimat în complementul față de doi

Total instrucțiuni : 4 * 1 = 4

Acțiunea :



Se testează cîte un bit al registrului F. Dacă condiția căutată este adevărată, se efectuează saltul (PC=PC+d). În caz contrar se execută instrucțiunea următoare. Semnificația condițiilor este :

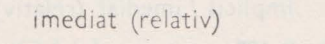
NZ — non zero → salt dacă Z=0
Z — zero → salt dacă Z=1
NC — non carry → salt dacă Cy=0
C — carry → salt dacă Cy=1

Necesar de memorie : 2 byte

Observație : În momentul în care se efectuează calculul adresei de salt, contorul program PC conține deja adresa instrucțiunii următoare, adresa care este cu +2 mai mare decît adresa instrucțiunii de salt însăși.
De aceea plaja de adrese ce poate fi acoperită prin JR c,d este de [—126, +129] față de adresa de început a instrucțiunii de salt.

Necesar de timp : 12 (4,3,5,) tacți procesor dacă condiția este adevărată
7 (4,3) tacți procesor dacă condiția este falsă (fără salt).

Structura instrucțiunii :



Tipuri de adresare : imediat (relativ)

Memento de cod : **JR c,d**

20_H	28_H	30_H	38_H
NZ	Z	NC	C

Flaguri afectate : nici unul

•	•	X	•	X	•	•	•
S	Z		H		P	N	Cy

Exemple de utilizare : vol. II, p. 145, p. 155

Clasa 8 : Instrucțiuni de salt

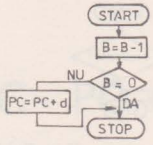
Grupa 6 : Salt condiționat relativ cu decrementare

Mnemonica : **DJNZ d** (Decrement Jump Not Zero)

unde : **d** este deplasamentul [—**128, +127**] exprimat în complementul față de doi

Total instrucțiuni : 1

Acțiunea :



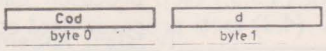
Conținutul registrului **B** este decrementat cu 1. Dacă astfel s-a ajuns la valoarea **B=0**, atunci se execută instrucțiunea următoare din program. În caz contrar se efectuează un salt relativ **(PC=PC+d)**.

Necesar de memorie : 2 byte

Observație : În momentul în care se efectuează calculul adresei de salt, contorul program **PC** conține deja adresa instrucțiunii următoare, adresă care este cu +2 mai mare decît adresa instrucțiunii de salt însăși. De aceea plaja de adrese ce poate fi acoperită prin **DJNZ d**, este de [—**126, +129**] față de adresa de început a instrucțiunii **DJNZ**.

Necesar de timp : 13 (5,3,5) tacți procesor dacă se efectuează saltul (B≠0)
8 (5,3) tacți procesor dacă nu se efectuează saltul (B=0)

Structura instrucțiunii :

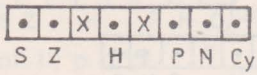


Tipuri de adresare : op1 / adresa de salt
implicit / imediat (relativ)

Memento de cod : DJNZ d

cod : 10H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 174 p. 176

Clasa 9 : Instrucţiuni de apel şi revenire din subrutine

Grupa 1 : Apel necondiţionat

Mnemonica : **CALL nn**

unde : nn este o adresă [0,65535]

Total instrucţiuni : 1

Acţiunea :

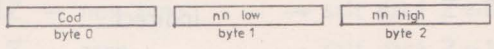
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{high}$$
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{low}$$
$$PC = nn$$

Conţinutul contorului program PC este salvat în vîrful stivei, după care PC este încărcat cu noua adresă nn. Conţinutul indicatorului de stivă SP este decrementat cu 2.

Necesar de memorie : 3 byte

Necesar de timp : 17 (4,3,4,3,3) tacţi procesor

Structura instrucţiuni :

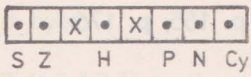


Tipuri de adresare : **adresa de salt / adresa de revenire**
direct / indirect

Memento de cod : CALL nn

cod : CDH

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 155, p. 173

Clasa 9 : Instrucțiuni de apel și revenire din subrutine

Grupa 2 : Revenire necondiționată

Mnemonica : **RET**

Total instrucțiuni : 1

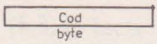
Acțiunea : $PC_{low} \longleftarrow (SP)$
$SP = SP + 1$
$PC_{high} \longleftarrow (SP)$
$SP = SP + 1$

Adresa de revenire din subrutină în programul apelant este presupusă a se afla în vîrful stivei. Ea se încarcă în contorul program **PC**, după care se efectuează saltul. Conținutul indicatorului de stivă **SP** este incrementat cu 2.

Necesar de memorie : 1 byte

Necesar de timp : 10 (4,3,3) tacți procesor

Structura instrucțiunii :

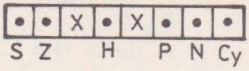


Tipuri de adresare : adresa de revenire
indirect

Memento de cod : RET

cod : C9H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 155, p. 176

Clasa 9 : Instrucţiuni de apel şi revenire din subrutine

Grupa 3 : Apel condiţionat

Mnemonica generică : CALL c,nn

unde : **c** este condiţia şi poate fi : **NZ, Z, NC, C, PO, PE, P** sau **M**
nn este o adresă **[0,65535]**

Total instrucţiuni : 8 * 1 = 8
Acţiunea : dacă **c** este adevărată se execută saltul la subrutină

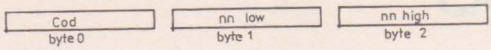
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{high}$$
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{low}$$
$$PC = nn$$

Se salvează adresa de revenire în vîrful stivei, se încarcă **PC** cu **nn**. Indicatorul de stivă **SP** este decrementat cu 2. Dacă **c** este falsă se execută instrucţiunea următoare din program.
Semnificaţia condiţiilor :

NZ	— non zero	→ salt dacă Z=0
Z	— zero	→ salt dacă Z=1
NC	— non carry	→ salt dacă Cy=0
C	— carry	→ salt dacă Cy=1
PO	— parity odd	→ salt dacă P/V=0
PE	— parity even	→ salt dacă P/V=1
P	— plus	→ salt dacă S=0
M	— minus	→ salt dacă S=1

Necesar de memorie : 3 byte

Necesar de timp : 17 (4,3,4,3,3) tacţi procesor dacă se efectuează saltul
10 (4,3,3) tacţi procesor dacă nu se efectuează saltul

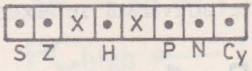
Structura instrucțiunii : 

Tipuri de adresare : adresa de salt / adresa de revenire
direct / indirect

Memento de cod : CALL c,nn

	C4 H	CC H	D4 H	DC H	E4 H	EC H	F4 H	FC H
CALL C	NZ	Z	NC	C	PØ	PE	P	M

Flaguri afectate : nici unul



exemple de utilizare : vol. II, p. 155

Clasa 9 : Instrucţiuni de apel şi revenire din subrutine

Grupa 4 : Revenire condiţionată

Mnemonica generică : **RET c**

unde : c este condiţia şi poate fi **NZ, Z, NC, C, PO, PE, P** sau **M**

Total instrucţiuni : 8 * 1 = 8

Acţiunea : dacă c este adevărată se execută revenirea în programul apelant

$$PC_{low} \leftarrow (SP)$$
$$SP = SP + 1$$
$$PC_{high} \leftarrow (SP)$$
$$SP = SP + 1$$

Adresa de revenire este presupusă a fi în vîrful stivei. Ea se încarcă în contorul program. Conţinutul indicatorului de stivă este incrementat cu 2.

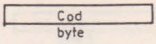
Dacă condiţia c este falsă se execută următoarea instrucţiune din program.

Semnificaţia condiţiilor :

NZ	— non zero	→ salt dacă Z=0
Z	— zero	→ salt dacă Z=1
NC	— non carry	→ salt dacă Cy=0
C	— carry	→ salt dacă Cy=1
PO	— parity odd	→ salt dacă P/V=0
PE	— parity even	→ salt dacă P/V=1
P	— plus	→ salt dacă S=0
M	— minus	→ salt dacă S=1

Necesar de memorie : 1 byte

Necesar de timp : 11 (5,3,3) tacţi procesor dacă se efectuează revenirea
5 (5) tacţi procesor dacă nu se efectuează revenirea

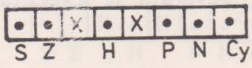
Structura instrucţiunii : 

Tipuri de adresare : adresa de revenire
indirect

Memento de cod : RET c

$C0_H$	$C8_H$	$D0_H$	$D8_H$	$E0_H$	$E8_H$	$F0_H$	$F8_H$
NZ	Z	NC	C	PØ	PE	P	M

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 182

Clasa 9 : Instrucţiuni de apel şi revenire din subrutine

Grupa 5 : Apel scurt, necondiţionat, la adrese fixe

Mnemonica generică : **RST n**

unde : **n** este o adresă din pagina 0 şi poate fi **0, 8, 10H, 18H, 20H, 28H, 30H** sau **38H**

Total instrucţiuni : $8 * 1 = 8$

Acţiunea :

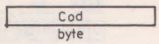
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{high}$$
$$SP = SP - 1$$
$$(SP) \leftarrow PC_{low}$$
$$PC = n$$

Se salvează adresa de revenire **(PC)** în vîrful stivei. **PC** se încarcă cu **n** şi se execută saltul. Conţinutul indicatorului de stivă **SP** este decrementat cu 2.

Necesar de memorie : 1 byte

Necesar de timp : 11 (5,3,3) tacţi procesor

Structura instrucţiunii :

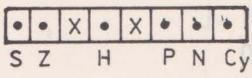


Tipuri de adresare : **adresa de salt / adresa de revenire**
implicit / indirect

Memento de cod : **RST n**

$C7_H$	CF_H	$D7_H$	DF_H	$E7_H$	EF_H	$F7_H$	FF_H
0	8	10H	18H	20H	28H	30H	38H

Flaguri afectate : nici unul



Exemple de utilizare :

Clasa 10 : Instrucțiuni de rotire/deplasare

Grupa 1 : Rotiri „cu" carry a regiștrilor

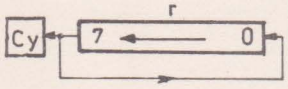
Mnemonica generică : R x C r (RLC sau RRC)

unde : x poate fi *litera* **L** sau **R**
r poate fi **B, C, D, E, H, L** sau **A**

Total instrucțiuni : $2 * 7 = 14$

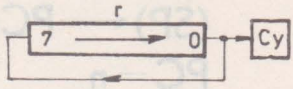
Acțiunea :

RLC



Conținutul registrului **r** este deplasat cu o poziție la **stînga.**
Bitul **7** se transfera în flagul **carry** și în bitul **0.**

RRC

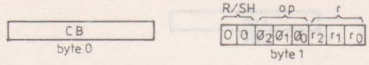


Conținutul registrului **r** este deplasat cu o poziție la **dreapta.**
Bitul **0** se transferă în flagul **carry** și în bitul **7.**

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacți procesor

Structura instrucțiunii :



unde : **o2o1o0** reprezintă codul pe 3 bit al operației RØT/SHIFT

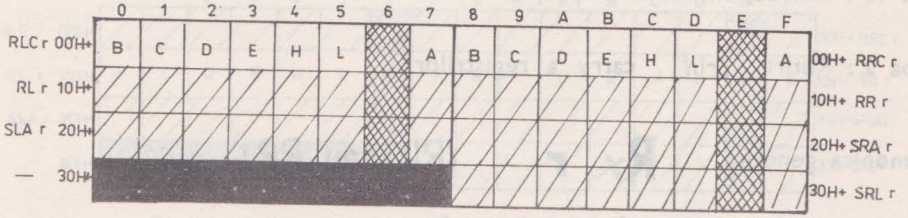
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

r2r1r0 reprezintă codul pe 3 bit al regiștrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

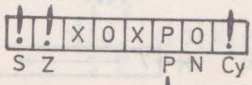
Tipuri de adresare : implicit

Memento de cod : R×C r (RLC sau RRC)



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.
Exemplu : RLC D 00H+2=02H

Flaguri afectate : toate



flagul P/V indică paritatea

Exemple de utilizare : vol. II, p. 198

Clasa 10 : Instrucţiuni de rotire/deplasare

Grupa 2 : Rotiri „prin" carry a regiştrilor

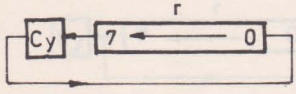
Mnemonica generică : **Rx r** **(RL sau RR)**

unde : x poate fi *litera* **L** sau **R**
r poate fi **B, C, D, E, H, L** sau **A**

Total instrucţiuni : $2 * 7 = 14$

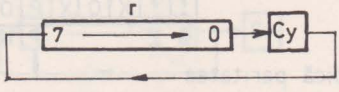
Acţiunea :

RL



Conţinutul registrului **r** este deplasat cu o poziţie la **stînga.** Bitul **7** se transferă în **carry ; carry** se transferă în bitul **0.**

RR

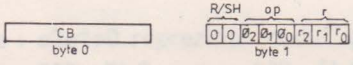


Conţinutul registrului **r** este deplasat cu o poziţie la **dreapta.** Bitul **0** se transferă în **carry ; carry** se transferă în bitul **7.**

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacţi procesor

Structura instrucţiunii :



unde : **o2o1o0** reprezintă codul pe 3 bit al operaţiei ROT/SHIFT

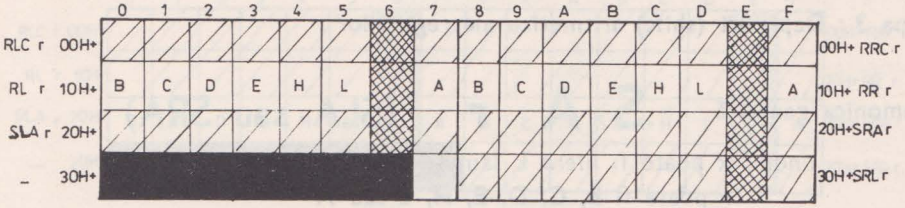
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

r2r1r0 reprezintă codul pe 3 bit al regiştrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

Tipuri de adresare : implicit

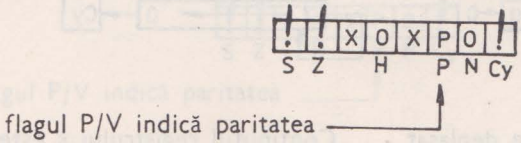
Memento de cod : R r (RL sau RR)



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.

Exemplu : RR L 10H+DH=1DH

Flaguri afectate : toate



Exemple de utilizare : vol II, p. 199

Clasa 10 : Instucţiuni de rotire/deplasare

Grupa 3 : Deplasări (shift) aritmetice ale regiştrilor

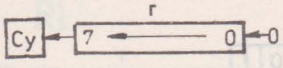
Mnemonica generică : **S x A r** (SLA sau SRA)

unde : x poate fi *litera* L sau R
r poate fi B, C, D, E, H, L sau A

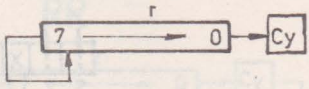
Total instrucţiuni : 2 * 7 = 14

Acţiunea :

SLA



SRA



Conţinutul registrului **r** este deplasat cu o poziţie la **stînga.** Bitul **7** se transferă în **carry.** În bitul **0** se inserează valoarea **0.**

Conţinutul registrului **r** este deplasat cu o pozţie la **dreapta.** Bitul **0** se transferă în **carry.** Bitul **7** rămîne **neschimbat.**

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacţi procesor

Structura instrucţiunii :

unde : **o2o1o0** reprezintă codul pe 3 bit al operaţiei ROT/SHIFT

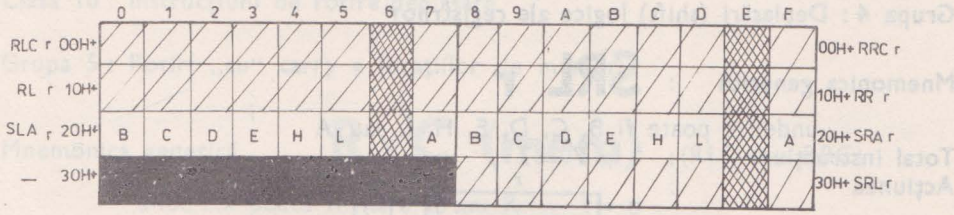
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

r2r1r0 reprezintă codul pe 3 bit al regiştrilor interni

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

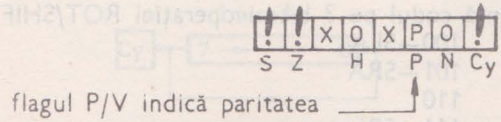
Tipuri de adresare : implicit

Memento de cod : S×A r (SLA sau SRA)



Codul dorit se obține însumînd numerele aferente rîndului și coloanei în care se află instrucțiunea căutată.
Exemplu: SRA A 20H + FH = 2FH

Flaguri afectate : toate



Exemple de utilizare :

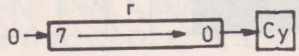
Clasa 10 : Instrucţiuni de rotire/deplasare
Grupa 4 : Deplasări (shift) logice ale regiştrilor

Mnemonica generică : **SRL r**

unde : r poate fi **B, C, D, E, H, L** sau **A**

Total instrucţiuni : 1 * 7 = 7

Acţiunea :

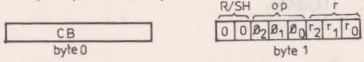


Conţinutul registrului **r** este deplasat cu o poziţie la **dreapta.** Bitul **0** se transferă în **carry.**
În bitul **7** se inserează valoarea **0.**

Necesar de memorie : 2 byte

Necesar de timp : 8 (4,4) tacţi procesor

Structura instrucţiunii :



unde : **o2o1o0** reprezintă codul pe 3 bit al operaţiei ROT/SHIFT

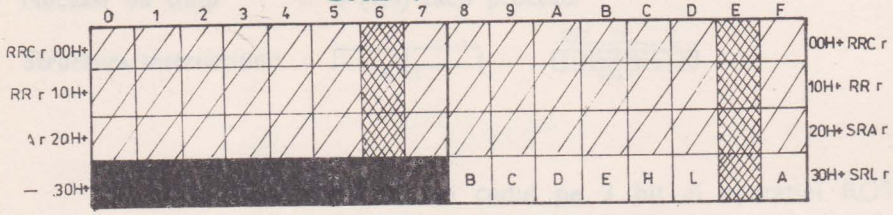
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

r2r1r0 reprezintă codul pe 3 bit al regiştrilor interni :

000—B	100—H
001—C	101—L
010—D	110
011—E	111—A

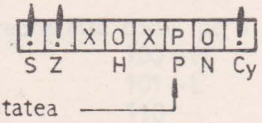
Tipuri de adresare : implicit

Memento de cod : **SRL r**



Codul dorit se obţine însumînd numerele aferente rîndului şi coloanei în care se află instrucţiunea căutată.
Exemplu : SRL E 30H+BH=3BH

Flaguri afectate : toate



Exemple de utilizare : vol. II, p. 197

Clasa 10 : Instrucţiuni de rotire/deplasare

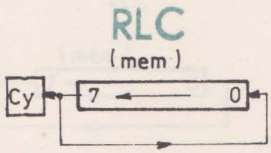
Grupa 5 : Rotiri „cu" carry a locaţiilor de memorie

Mnemonica generică : **R × C (mem)** (RLC sau RRC)

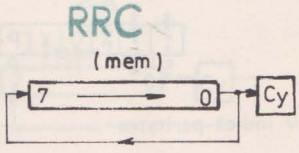
unde : x poate fi *litera* **L** sau **R**
mem poate fi **HL, IX+IND** sau **IY+IND**

Total instrucţiuni : 2 * 3 = 6

Acţiunea :



Conţinutul celulei de memorie adresată prin **mem** este deplasat cu o poziţie la **stînga.**
Bitul **7** se transferă în **carry** şi în bitul **0.**

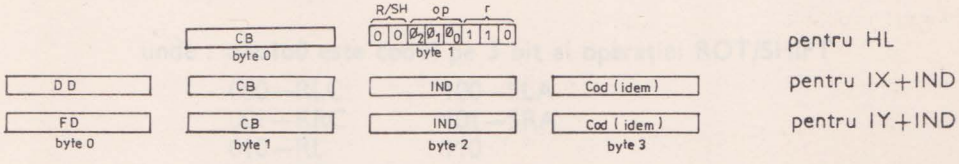


Conţinutul celulei de memorie adresate prin **mem** este deplasat cu o poziţie la **dreapta.**
Bitul **0** se transferă în **carry** şi în bitul **7**

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacţi procesor pentru HL
23 (4,4,3,5,4,3) tacţi procesor pentru IX+IND sau IY+IND

Structura instrucţiunii :



unde : o2o1o0 este codul pe 3 bit al operației ROT/SHIFT

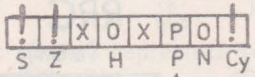
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : R C (mem) (RLC sau RRC)

	RLC (mem)	RRC (mem)
cod :	06H	0EH

Flaguri afectate : toate



flagul P/V indică paritatea

Exemple de utilizare :

Clasa 10 : Instrucţiuni de rotire/deplasare

Grupa 6 : Rotiri „prin" carry ale locaţiilor de memorie

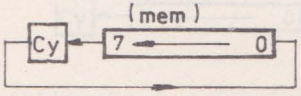
Mnemonica generică : **R x (mem)** (RL sau RR)

unde : x poate fi *litera* L sau R
mem poate fi **HL, IX+IND** sau **IY+IND**

Total instrucţiuni : 2 * 3 = 6

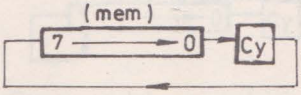
Acţiunea :

RL



Conţinutul celulei de memorie adresată prin **mem** este deplasat la **stînga** cu o poziţie. Bitul 7 se transferă în **carry**. **Carry** se transferă în bitul **0**.

RR

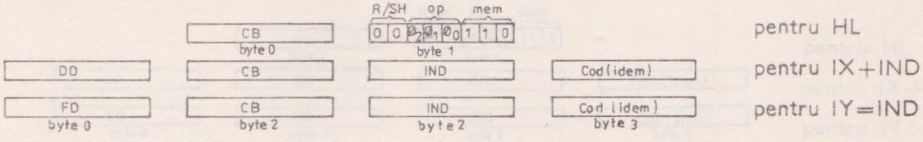


Conţinutul celulei de memorie adresată prin **mem** este deplasat la **dreapta** cu o poziţie. Bitul **0** se transferă în **carry**. **Carry** se transferă în bitul **7**.

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacţi procesor pentru HL
23 (4,4,3,5,4,3) tacţi procesor pentru IX+IND sau IY+IND

Structura instrucţiunii :



unde : **o2o1o0** este codul pe 3 bit al operaţiei ROT/SHIFT

000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

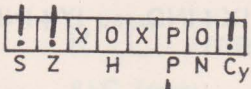
Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : R× (mem) (RL sau RR)

RL (mem) RR (mem)

cod : 16H 1EH

Flaguri afectate : toate



flagul P/V indică paritatea

Exemple de utilizare :

Clasa 10 : Instrucţiuni de rotire/deplasare

Grupa 7 : Deplasări (shift) aritmetice ale locaţiilor de memorie

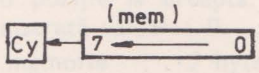
Mnemonica generică : S × A (mem) (SLA sau SRA)

unde : x poate fi *litera* **L** sau **R**
mem poate fi **HL**, **IX+IND** sau **IY+IND**

Total instrucţiuni : 2 * 3 = 6

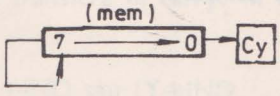
Acţiunea :

SLA



Conţinutul locaţiei de memorie adresată prin **mem** este deplasat la **stînga** cu o poziţie. Bitul **7** se transferă în **carry**. În bit **0** se inserează valoarea **0**.

SRA

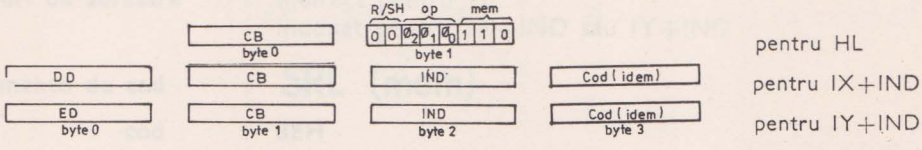


Conţinutul locaţiei de memorie adresată prin **mem** este deplasat la **dreapta** cu o poziţie. Bitul **0** se transferă în **carry**. Bitul **7** rămîne **neschimbat.**

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacţi procesor pentru HL
23 (4,4,3,5,4,3) tacţi procesor pentru IX+IND sau IY+IND

Structura instrucţiunii :



unde : **o2o1o0** este codul pe 3 bit al operaţiei ROT/SHIFT

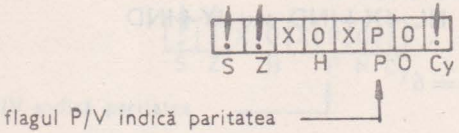
000—RLC	100—SLA
001—RRC	101—SRA
010—RL	110
011—RR	111—SRL

Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : S×A (mem) (SLA sau SRA)

	SLA (mem)	SRA (mem)
cod :	26H	2EH

Flaguri afectate : toate



Exemple de utilizare :

Clasa 10 : Instrucțiuni de rotire/deplasare

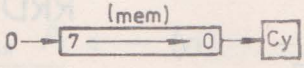
Grupa 8 : Deplasări (shift) logice ale locațiilor de memorie

Mnemonica generică : **SRL (mem)**

unde : mem poate fi **HL, IX+IND** sau **IY+IND**

Total instrucțiuni : 1 * 3 = 3

Acțiunea : **SRL**

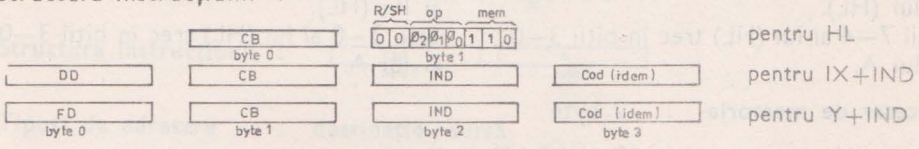


Conținutul celulei de memorie adresată prin **mem** este deplasat cu o poziție la **dreapta**. Bitul **0** se transferă în **carry**. În bitul **7** este inserată valoarea **0**.

Necesar de memorie : 2 byte pentru HL
4 byte pentru IX+IND sau IY+IND

Necesar de timp : 15 (4,4,4,3) tacți procesor pentru HL
23 (4,4,3,5,4,3) tacți procesor pentru IX+IND sau IY+IND

Structura instrucțiunii :



unde : **o2o1o0** este codul pe 3 bit al operației ROT/SHIFT

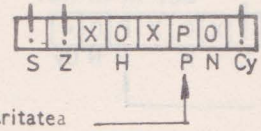
000—RLC	100—SLA
001—RRC	101—SRA
010—RC	110
011—RR	111—SRL

Tipuri de adresare : indirect pentru HL
indexat pentru IX+IND sau IY+IND

Memento de cod : **SRL (mem)**

cod : **3EH**

Flaguri afectate : toate



flagul P/V indică paritatea

Exemple de utilizare :

Clasa 10 : Instrucţiuni de rotire/deplasare

Grupa 9 : Rotirea digiţilor de 4 bit

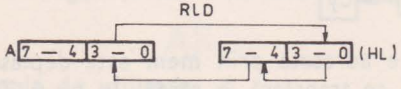
Mnemonica generică : **R×D (RLD sau RRD)**

unde : x poate fi *litera* L sau R

Total instrucţiuni : 1 + 1 = 2

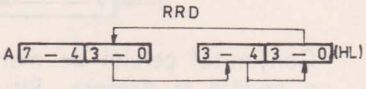
Acţiunea

RLD



Conţinutul celulei de memorie adresată prin conţinutul registrului dublu HL, este rotit la stînga folosind digitul inferior al registrului A.
Biţii 3—0 ai lui A trec în biţii 3—0 ai lui (HL).
Biţii 3—0 al lui (HL) trec în biţii 7—4 ai lui (HL).
Biţii 7—4 ai lui (HL) trec în biţii 3—0 ai lui A.

RRD

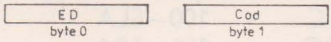


Conţinutul celulei de memorie adresată prin conţinutul registrului dublu HL, este rotit la dreapta folosind digitul inferior al registrului A.
Biţii 3—0 ai lui A trec în biţii 7—4 ai lui (HL).
Biţii 7—4 ai lui (HL) trec în biţii 3—0 ai lui (HL).
Biţii 3—0 ai lui (HL) trec în biţii 3—0 ai lui A.

Necesar de memorie : 2 byte

Necesar de timp : 18 (4,4,3,4,3) tacţi procesor

Structura instrucţiunii :



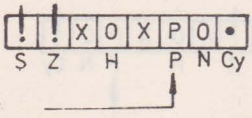
Tipuri de adresare : op1 / op2
implicit / indirect

Memento de cod : **R×D (RLD sau RRD)**

RLD RRD

cod : 6FH 67H

Flaguri afectate : toate cu excepţia lui carry



Exemple de utilizare : vol. II, p. 186

Clasa 11 : Instrucţiuni de intrare/ieşire

Grupa 1 : Intrare (Input) cu adresare directă

Mnemonica generică : **IN A,(n)**

unde : n este o adresă de port [0,255].

Total instrucţiuni : 1

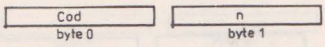
Acţiunea : A ⟵ (n)

Conţinutul portului de intrare adresat prin n este transferat în registrul A. Conţinutul vechi al acumulatorului A se pierde.

Necesar de memorie : 2 byte

Necesar de timp : 11 (4,3,4) tacţi procesor
În ultimul ciclu maşină, **M3** (4 tacţi procesor), starea magistralei de adrese este următoarea :
A0—A7 numărul n
A8—A15 vechiul conţinut al registrului A

Structura instrucţiunii :

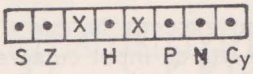


Tipuri de adresare : **destinaţie / sursă**
implicit / direct

Memento de cod : **IN A,(n)**

cod : DBH

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 158

Clasa 11 : Instrucțiuni de intrare/ieșire

Grupa 2 : Intrare (Input) cu adresare indirectă

Mnemonica generică : **IN r,(C)**

unde : r poate fi B, C, D, E, H, L, sau A

Total instrucțiuni : 7 * 1 = 7

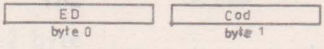
Acțiunea : r ⟵ (C)

Conținutul portului de intrare adresat prin conținutul registrului C este transferat în registrul r. Vechiul conținut al registrului r se pierde.

Necesar de memorie : 2 byte

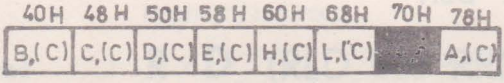
Necesar de timp : 12 (4,4,4) tacți procesor
În ultimul ciclu mașină, M3 (4 tacți procesor), starea magistralei de adrese este următoarea :
A0—A7 conținutul registrului C
A8—A15 conținutul registrului B

Structura instrucțiunii :

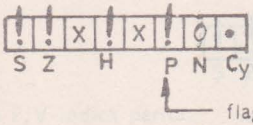


Tipuri de adresare : destinație / sursă
implicit / indirect

Memento de cod : IN r,(C)



Flaguri afectate : toate cu excepția lui carry
Se observă deosebirea față de input cu adresare directă, care nu poziționează indicatorii de stare.



Exemple de utilizare : vol. II, p. 197

Clasa 11 : Instrucțiuni de intrare/ieșire

Grupa 3 : Ieșire (Output) cu adresare directă

Mnemonica generică : **OUT (n),A**

unde : n este o adresă de port [0,255].

Total instrucțiuni : 1

Acțiunea : **(n) ⟵ A**

Conținutul acumulatorului este transferat în portul de ieșire cu adresa n. Conținutul acumulatorului nu este afectat.

Necesar de memorie : 2 byte

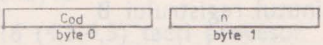
Necesar de timp : 11 (4,3,4) tacți procesor

În ultimul ciclu mașină, **M3** (4 tacți procesor), starea magistralei de adrese este următoarea :

A0—A7 conține numărul n

A8—A15 conținutul registrului A

Structura instrucțiunii :

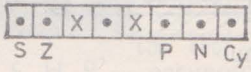


Tipuri de adresare : destinație / sursă
direct / implicit

Memento de cod : **OUT (n),A**

cod : D3H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 145

Clasa 11 : Instrucțiuni de intrare/ieșire

Grupa 4 : Ieșire (output) cu adresare indirectă

Mnemonica generică : **OUT (C),r**

unde : r poate fi B, C, D, E, H, L sau A

Total instrucțiuni : 7 * 1 = 7

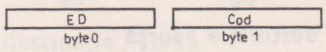
Acțiunea : (C) ⟵ r

Conținutul registrului r este transferat la portul de ieșire avînd adresa specificată prin conținutul registrului C. Conținutul registrului r nu se modifică.

Necesar de memorie : 2 byte

Necesar de timp : 12 (4,4,4) tacți procesor
În ultimul ciclu mașină, M3 (4 tacți procesor), starea magistralei de adrese este următoarea :
A0—A7 conținutul registrului C
A8—A15 conținutul registrului B

Structura instrucțiunii :

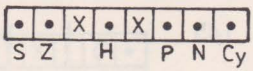


Tipuri de adresare : destinație / sursă
indirect / implicit

Memento de cod : OUT (C),r

41H	49H	51H	59H	61H	69H	71H	79H
(C),B	(C),C	(C),D	(C),E	(C),H	(C),L		(C),A

Flaguri afectate : nici unul



Exemple de utilizare :

Clasa 11 : **Instrucţiuni de intrare/ieşire**

Grupa 5 : **I/E blocuri de date — Intrare (input) simplă**

Mnemonica generică : **IN × (INI sau IND)**

unde : x poate fi *litera* **I** sau **D**

Total instrucţiuni : 1 + 1 = 2

Acţiunea

INI (in&increment)

(HL) ← (C)
HL = HL + 1
B = B − 1

Conţinutul portului de intrare adresat prin conţinutul registrului **C**, este transferat în memorie la adresa specificată de registrul dublu **HL**. Indicatorul de adresă **HL** este incrementat cu 1. Numărătorul de octeţi **B** este decrementat cu 1.

IND (in&decrement)

(HL) ← (C)
HL = HL − 1
B = B − 1

Idem cu **INI**, dar indicatorul de adrese este decrementat cu 1.

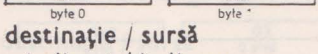
Necesar de memorie : 2 byte

Necesar de timp : 16 (4,5,4,3) tacţi procesor

În ultimul ciclu maşină **M3** (4 tacţi) starea magistralei de adrese este următoarea :

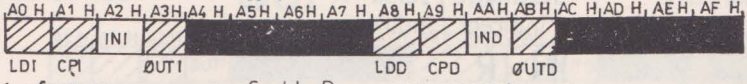
A0 — A7 conţinutul registrului **C**
A8 — A15 conţinutul registrului **B decrementat**

Structura instrucţiunii :

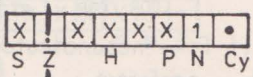


Tipuri de adresare : **destinaţie / sursă**
indirect / indirect

Memento de cod : **IN ×**



Flaguri afectate : S, H, P — necunoscute
N — devine 1
Cy — neafectat



Z=1 dacă după execuţie B=0
Z=0 dacă după execuţie B≠0

Exemple de utilizare :

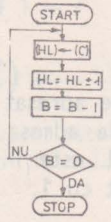
Clasa 11 : Instrucțiuni de intrare/ieșire
Grupa 6 : I/E blocuri de date — Intrare (input) multiplă

Mnemonica generică : **IN×R** (INIR sau INDR)

unde : x poate fi *litera* I sau D

Total instrucțiuni : 1 + 1 = 2

Acțiunea :



} + INIR — input, increment, repeat
} — INDR — input, decrement, repeat

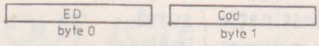
Conținutul portului de intrare adresat prin conținutul registrului **C**, este transferat în celula de memorie adresată prin registrul dublu **HL**. Indicatorul de adresă **HL** este *incrementat* **(INIR)** sau *decrementat* **(INDR)** cu 1. Numărătorul de octeți **B** este decrementat cu 1. Operația se repetă pînă cînd **B=0**.

Necesar de memorie : 2 byte

Necesar de timp : 21 (4,5,4,3,5) tacți procesor dacă după execuție B≠0
16 (4,5,4,3) tacți procesor dacă după execuție B=0

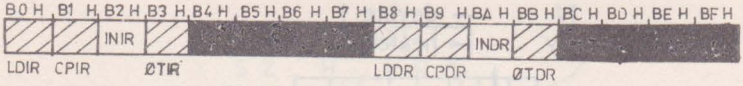
În ciclul de intrare, **M3** (4 tacți procesor), starea magistralei de adrese este următoarea :
A0 — A7 conținutul registrului **C**
A8 — A15 conținutul registrului **B decrementat**

Structura instrucțiunii :



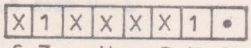
Tipuri de adresare : **destinație / sursă**
indirect / indirect

Memento de cod : **IN×R**



Flaguri afectate : S, H, P — necunoscute Z și N — devin 1
Cy — neafectat

Exemple de utilizare :



Clasa 11 : Instrucțiuni de intrare/ieșire
Grupa 7 : I/E blocuri de date — ieșire (output) simplă

Mnemonica generică : **OUT ×** (OUTI sau OUTD)

unde : × poate fi *litera* I sau D

Total instrucțiuni : 1 + 1 = 2

Acțiunea : **OUTI** (out&increment)

(C) ← (HL)
HL = HL + 1
B = B − 1

Conținutul celulei de memorie adresată prin registrul dublu **HL** este transferat la portul de ieșire adresat prin conținutul registrului **C**. Indicatorul de adresă **HL** este incrementat cu 1. Numărătorul de octeți **B** este decrementat cu 1.

OUTD (out&decrement)

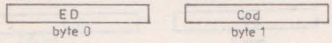
(C) ← (HL)
HL = HL + 1
B = B − 1

Idem cu **OUTI**, dar cu deosebirea că indicatorul de adresă **HL** este decrementat cu 1.

Necesar de memorie : 2 byte
Necesar de timp : 16 (4,5,3,4) tacți procesor

În timpul ciclului out, **M4** (4 tacți procesor), starea magistralei de adrese este următoarea :
A0 — A7 conținutul registrului **C**
A8 — A15 conținutul registrului **B decrementat**

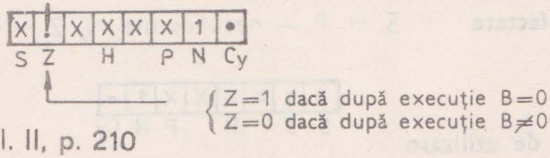
Structura instrucțiunii :



Tipuri de adresare : destinație / sursă
indirect / indirect

Memento de cod : **OUT ×**

Flaguri afectate : S, H, P — necunoscute, N — devine 1, Cy — neafectat



Exemple de utilizare : vol. II, p. 210

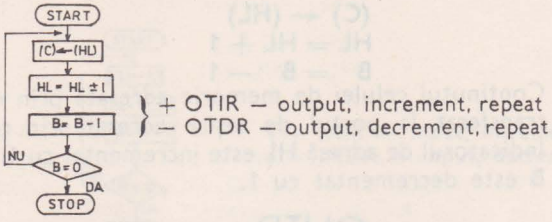
Clasa 11 : Instrucţiuni de intrare/ieşire
Grupa 8 : I/E blocuri de date — ieşire (output) multiplă

Mnemonica generică : **OT×R** (OTIR sau OTDR)

unde : x poate fi *litera* I sau D

Total instrucţiuni : 1 + 1 = 2

Acţiunea :



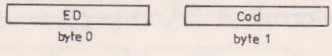
Conţinutul celulei de memorie adresată prin registrul dublu **HL** este transferat în portul de ieşire adresat prin conţinutul registrului **C**. Indicatorul de adresă HL este *incrementat* **(OTIR)** sau *decrementat* **(OTDR)** cu 1. Numărătorul de octeţi este decrementat cu 1. Operaţia se reia pînă cînd **B=0.**

Necesar de memorie : 2 byte

Necesar de timp : 21 (4,5,3,4,5) tacţi procesor dacă B≠0
16 (4,5,3,4) tacţi procesor dacă B=0

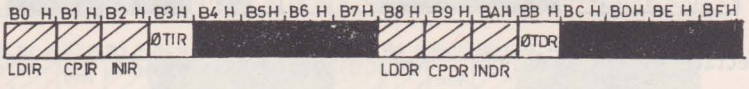
În ciclul out, **M4** (4 tacţi procesor), starea magistralei de adrese este următoarea :
A0 — A7 conţinutul registrului **C**
A8 — A15 conţinutul registrului **B decrementat**

Structura instrucţiunii

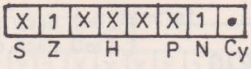


Tipuri de adresare : **destinaţie / sursă**
indirect / indirect

Memento de cod : **OT×R**



Flaguri afectate : S, H, P — necunoscute N, Z — devin 1 Cy — neafectat



Exemple de utilizare :

Clasa 12 : Instrucţiuni de comandă

Grupa 1 : Instrucţiunea HALT

Mnemonica : **HALT**

Total instrucţiuni : 1

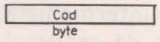
Acţiunea :

După execuţia acestei instrucţiuni microprocesorul „se opreşte". Se execută în continuare fetch-uri (care nu se iau în considerare) pentru a se asigura reîmprospătarea memoriei dinamice, dacă este cazul. Ieşirea din această stare nu se poate face decît prin recepţia unei întreruperi sau a unui semnal de RESET.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Structura instrucţiunii :

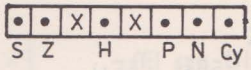


Tipuri de adresare : —

Memento de cod : HALT

cod : 76H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 145

Clasa 12 : Instrucțiuni de comandă **SYS**

Grupa 2 : Stabilirea modului de întrerupere

Mnemonica generică : **IM x**

unde : x poate fi 0, 1 sau 2

Total instrucțiuni : $3 * 1 = 3$

Acțiunea :

Stabilește modul de întrerupere specificat :

IM 0

— în acest mod dispozitivul care cere întreruperea va depune în ciclul de acceptare a întreruperii (interrupt acknowledge) pe magistrala de date codul unei instrucțiuni (de obicei CALL sau RST). Dacă instrucțiunea astfel forțată are mai mulți octeți, dispozitivul întrerupător îi va livra și pe aceștia în cicluri mașină succesive.

IM 1

— în acest mod microrpocesorul execută automat o instrucțiune RST 38H la acceptarea unei întreruperi.

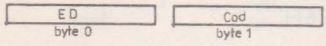
IM 2

— dispozitivul care cere întreruperea furnizează un octet care va fi octetul inferior intr-o adresă intermediară. Octetul superior al adresei intermediare este furnizat de registrul I. Citind cele două celule de memorie succesive, începînd cu adresa specificată prin adresa intermediară, se obține adresa de salt la care se regăsește rutina de tratare a întreruperii respective.

Necesar de memorie : 2 byte

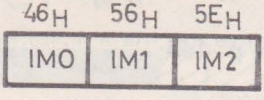
Necesar de timp : 8 (4,4) tacți procesor

Structura instrucțiunii :

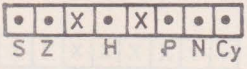


Tipuri de adresare : implicit

Memento de cod : **IM x**



Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 146

Clasa 12 : Instrucţiuni de comandă

Grupa 3 : Validarea şi inhibarea sistemului de întreruperi

Mnemonica generică : x**I** (EI sau DI)

unde : x poate fi *litera* E sau D

Total instrucţiuni : 1 + 1 = 2

Acţiunea :

EI

IFF1 = IFF2 = 1

Sistemul de întreruperi se validează. El va fi apt să accepte o întrerupere după execuţia primei instrucţiuni care urmează după **EI**. Starea validată durează pînă la execuţia primei instrucţiuni DI sau pînă la acceptarea primei cereri de întrerupere mascabilă.

DI

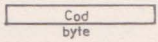
IFF1 = IFF2 = 0

Sistemul de întrerupere se inhibă. Nu se mai acceptă cererile de întrerupere mascabilă ci doar cele nemascabile. Starea de inhibare poate fi suspendată prin execuţia unei instrucţiuni **EI**.

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacţi procesor

Structura instrucţiunii :

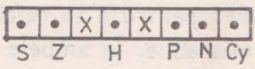


Tipuri de adresare : implicit

Memento de cod : x**I** (EI sau DI)

	EI	DI
cod :	FBH	F3H

Flaguri afectate : nici unul



Exemple de utilizare : vol. II, p. 149, p. 210

Clasa 12 : Instrucţiuni de comandă

Grupa 4 : Revenirea din rutinele de tratare a întreruperilor

Mnemonica generică : RET x (RETI sau RETN)

unde : x poate fi *litera* I sau N

Total instrucţiuni : 1 + 1 = 2

Acţiunea :

RETI

PClow ← (SP)
SP = SP + 1
PChigh ← (SP)
SP = SP + 1
! periferic Zilog

Din vîrful stivei **(SP)** se încarcă adresa de revenire în contorul program **PC.** Circuitele periferice din familia Zilog sesizează această instrucţiune pentru a-şi actualiza lista de priorităţi într-un sistem cu întreruperi multiple. (Este necesar a se executa instrucţiunea **EI înainte** de **RETI,** pentru a se revalida întreruperile mascabile).

RETN

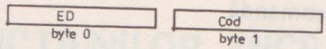
PClow ← (SP)
SP = SP + 1
PChigh ← (SP)
SP = SP + 1
IFF1 ← IFF2

Din vîrful stivei **(SP)** se încarcă adresa de revenire în contorul program **PC.** Conţinutul bistabilului **IFF2** care a fost incărcat cu **IFF1,** se copiază înapoi în **IFF1** pentru a se reface starea validată sau inhibată a întreruperilor mascabile. În ambele cazuri conţinutul indicatorului de stivă SP este incrementat cu 2.

Necesar de memorie : 2 byte

Necesar de timp : 14 (4,4,3,3) tacți procesor

Structura instrucțiunii :



Tipuri de adresare : adresa de revenire
indirect

Memento de cod : RET (RETI sau RETN)

	RETI	RETN
cod :	4DH	45H

Flaguri afectate : nici unul

•	•	X	•	X	•	•	•
S	Z		H		P	N	Cy

Exemple de utilizare : vol. II, p. 210

Clasa 12 : Instrucțiuni de comandă

Grupa 5 : Instrucțiunea NOP

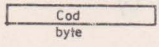
Mnemonica : **NOP**

Total instrucțiuni : 1

Acțiunea : PC = PC + 1

Necesar de memorie : 1 byte

Necesar de timp : 4 (4) tacți procesor

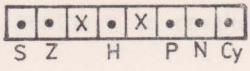
Structura instrucțiunii : 

Tipuri de adresare : —

Memento de cod : NOP

cod : 00H

Flaguri afectate : nici unul



Instrucțiunea NOP poate fi utilizată pentru a realiza temporizări elementare.

Exemple de utilizare :

F

LISTA INSTRUCŢIUNILOR MICROPROCESORULUI Z80

Prezentul capitol constă din 2 liste care cuprind întregul set de instrucţiun al microprocesorului **Z80** : prima în ordinea alfabetică a mnemonicelor, a doua ordonată crescător după cod.

Listele cuprind următoarele informaţii menţionate în antetul fiecărei pagini :

— **număr curent**
— **codul instrucţiunii**
— **mnemonica instrucţiunii**
— **indicatorii de condiţii (FLAG)**
— **numărul de cicluri maşină**
— **numărul de tacţi procesor şi repartizarea lor pe cicluri maşină**
— **timpul de execuţie** al instrucţiunii în μs pentru un microprocesor care funcţionează cu un tacţi de 2,5 MHz.

În cîmpul de comentarii se indică pagina la care se găseşte descrierea detaiată a instrucţiunii.

Notaţii utilizate pentru flaguri :

! — indicatorul este afectat conform rezultatului operaţiei
• — indicatorul nu e modificat de operaţie
0 — indicatorul e forţat în zero
1 — indicatorul e forţat în unu
X — indicatorul e indiferent
V — indicatorul P/V e poziţionat în conformitate cu depăşirea rezultatului
P — indicatorul P/V e poziţionat în conformitate cu paritatea rezultatului
I — conţinutul bistabilului de întreruperi IFF1

Pentru operanzi ai instrucţiunilor s-au folosit următoarele simboluri :

N — un octet cu valoarea cuprinsă între 00H — 0FFH
NN — doi octeţi cu valoarea cuprinsă între 0000H — 0FFFFH
IND — deplasament ce se adună la conţinutul unui registru index pentru obţinerea adresei locaţiei cu care lucrează instrucţiunea respectivă
DIS — deplasament ce se adună la adresa de asamblare a instrucţiunii respective pentru a obţine adresa de salt.

L-A-1

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
1	8E	ADC A,(HL)	!!X!XVO!	2	7 (4,3)	2.80	
2	DD 8E 05	ADC A,(IX+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
3	FD 8E 05	ADC A,(IY+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
4	8F	ADC A,A	!!X!XVO!	1	4 (4)	1.60	
5	88	ADC A,B	!!X!XVO!	1	4 (4)	1.60	
6	89	ADC A,C	!!X!XVO!	1	4 (4)	1.60	
7	8A	ADC A,D	!!X!XVO!	1	4 (4)	1.60	
8	8B	ADC A,E	!!X!XVO!	1	4 (4)	1.60	
9	8C	ADC A,H	!!X!XVO!	1	4 (4)	1.60	
10	8D	ADC A,L	!!X!XVO!	1	4 (4)	1.60	
11	CE 20	ADC A,N	!!X!XVO!	2	7 (4,3)	2.80	
12	ED 4A	ADC HL,BC	!!X!XVO!	4	15 (4,4,4,3)	6.00	
13	ED 5A	ADC HL,DE	!!X!XVO!	4	15 (4,4,4,3)	6.00	
14	ED 6A	ADC HL,HL	!!X!XVO!	4	15 (4,4,4,3)	6.00	
15	ED 7A	ADC HL,SP	!!X!XVO!	4	15 (4,4,4,3)	6.00	
16	86	ADD A,(HL)	!!X!XVO!	2	7 (4,3)	2.80	
17	DD 86 05	ADD A,(IX+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
18	FD 86 05	ADD A,(IY+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
19	87	ADD A,A	!!X!XVO!	1	4 (4)	1.60	
20	80	ADD A,B	!!X!XVO!	1	4 (4)	1.60	
21	81	ADD A,C	!!X!XVO!	1	4 (4)	1.60	
22	82	ADD A,D	!!X!XVO!	1	4 (4)	1.60	
23	83	ADD A,E	!!X!XVO!	1	4 (4)	1.60	
24	84	ADD A,H	!!X!XVO!	1	4 (4)	1.60	
25	85	ADD A,L	!!X!XVO!	1	4 (4)	1.60	
26	C6 20	ADD A,N	!!X!XVO!	2	7 (4,3)	2.80	
27	09	ADD HL,BC	..X!X.O!	3	11 (4,4,3)	4.40	
28	19	ADD HL,DE	..X!X.O!	3	11 (4,4,3)	4.40	
29	29	ADD HL,HL	..X!X.O!	3	11 (4,4,3)	4.40	
30	39	ADD HL,SP	..X!X.O!	3	11 (4,4,3)	4.40	
31	DD 09	ADD IX,BC	..X!X.O!	4	15 (4,4,4,3)	6.00	
32	DD 19	ADD IX,DE	..X!X.O!	4	15 (4,4,4,3)	6.00	
33	DD 29	ADD IX,IX	..X!X.O!	4	15 (4,4,4,3)	6.00	
34	DD 39	ADD IX,SP	..X!X.O!	4	15 (4,4,4,3)	6.00	
35	FD 09	ADD IY,BC	..X!X.O!	4	15 (4,4,4,3)	6.00	
36	FD 19	ADD IY,DE	..X!X.O!	4	15 (4,4,4,3)	6.00	
37	FD 29	ADD IY,IY	..X!X.O!	4	15 (4,4,4,3)	6.00	
38	FD 39	ADD IY,SP	..X!X.O!	4	15 (4,4,4,3)	6.00	
39	A6	AND (HL)	!!X1XPOO	2	7 (4,3)	2.80	
40	DD A6 05	AND (IX+IND)	!!X1XPOO	5	19 (4,4,3,5,3)	7.60	
41	FD A6 05	AND (IY+IND)	!!X1XPOO	5	19 (4,4,3,5,3)	7.60	
42	A7	AND A	!!X1XPOO	1	4 (4)	1.60	
43	A0	AND B	!!X1XPOO	1	4 (4)	1.60	
44	A1	AND C	!!X1XPOO	1	4 (4)	1.60	
45	A2	AND D	!!X1XPOO	1	4 (4)	1.60	
46	A3	AND E	!!X1XPOO	1	4 (4)	1.60	
47	A4	AND H	!!X1XPOO	1	4 (4)	1.60	
48	A5	AND L	!!X1XPOO	1	4 (4)	1.60	
49	E6 20	AND N	!!X1XPOO	2	7 (4,3)	2.80	
50	CB 46	BIT O,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
51	DD CB 05 46	BIT O,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
52	FD CB 05 46	BIT O,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
53	CB 47	BIT O,A	X!X1XXO.	2	8 (4,4)	3.20	
54	CB 40	BIT O,B	X!X1XXO.	2	8 (4,4)	3.20	
55	CB 41	BIT O,C	X!X1XXO.	2	8 (4,4)	3.20	
56	CB 42	BIT O,D	X!X1XXO.	2	8 (4,4)	3.20	
57	CB 43	BIT O,E	X!X1XXO.	2	8 (4,4)	3.20	
58	CB 44	BIT O,H	X!X1XXO.	2	8 (4,4)	3.20	
59	CB 45	BIT O,L	X!X1XXO.	2	8 (4,4)	3.20	
60	CB 4E	BIT 1,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
61	DD CB 05 4E	BIT 1,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
62	FD CB 05 4E	BIT 1,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
63	CB 4F	BIT 1,A	X!X1XXO.	2	8 (4,4)	3.20	
64	CB 48	BIT 1,B	X!X1XXO.	2	8 (4,4)	3.20	
65	CB 49	BIT 1,C	X!X1XXO.	2	8 (4,4)	3.20	
66	CB 4A	BIT 1,D	X!X1XXO.	2	8 (4,4)	3.20	
67	CB 4B	BIT 1,E	X!X1XXO.	2	8 (4,4)	3.20	
68	CB 4C	BIT 1,H	X!X1XXO.	2	8 (4,4)	3.20	
69	CB 4D	BIT 1,L	X!X1XXO.	2	8 (4,4)	3.20	
70	CB 56	BIT 2,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
71	DD CB 05 56	BIT 2,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
72	FD CB 05 56	BIT 2,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
73	CB 57	BIT 2,A	X!X1XXO.	2	8 (4,4)	3.20	
74	CB 50	BIT 2,B	X!X1XXO.	2	8 (4,4)	3.20	
75	CB 51	BIT 2,C	X!X1XXO.	2	8 (4,4)	3.20	
76	CB 52	BIT 2,D	X!X1XXO.	2	8 (4,4)	3.20	
77	CB 53	BIT 2,E	X!X1XXO.	2	8 (4,4)	3.20	
78	CB 54	BIT 2,H	X!X1XXO.	2	8 (4,4)	3.20	
79	CB 55	BIT 2,L	X!X1XXO.	2	8 (4,4)	3.20	
80	CB 5E	BIT 3,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
81	DD CB 05 5E	BIT 3,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
82	FD CB 05 5E	BIT 3,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
83	CB 5F	BIT 3,A	X!X1XXO.	2	8 (4,4)	3.20	
84	CB 58	BIT 3,B	X!X1XXO.	2	8 (4,4)	3.20	
85	CB 59	BIT 3,C	X!X1XXO.	2	8 (4,4)	3.20	
86	CB 5A	BIT 3,D	X!X1XXO.	2	8 (4,4)	3.20	
87	CB 5B	BIT 3,E	X!X1XXO.	2	8 (4,4)	3.20	
88	CB 5C	BIT 3,H	X!X1XXO.	2	8 (4,4)	3.20	
89	CB 5D	BIT 3,L	X!X1XXO.	2	8 (4,4)	3.20	
90	CB 66	BIT 4,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
91	DD CB 05 66	BIT 4,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
92	FD CB 05 66	BIT 4,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
93	CB 67	BIT 4,A	X!X1XXO.	2	8 (4,4)	3.20	
94	CB 60	BIT 4,B	X!X1XXO.	2	8 (4,4)	3.20	
95	CB 61	BIT 4,C	X!X1XXO.	2	8 (4,4)	3.20	
96	CB 62	BIT 4,D	X!X1XXO.	2	8 (4,4)	3.20	
97	CB 63	BIT 4,E	X!X1XXO.	2	8 (4,4)	3.20	
98	CB 64	BIT 4,H	X!X1XXO.	2	8 (4,4)	3.20	
99	CB 65	BIT 4,L	X!X1XXO.	2	8 (4,4)	3.20	
100	CB 6E	BIT 5,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
101	DD CB 05 6E	BIT 5,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
102	FD CB 05 6E	BIT 5,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
103	CB 6F	BIT 5,A	X!X1XXO.	2	8 (4,4)	3.20	
104	CB 68	BIT 5,B	X!X1XXO.	2	8 (4,4)	3.20	
105	CB 69	BIT 5,C	X!X1XXO.	2	8 (4,4)	3.20	
106	CB 6A	BIT 5,D	X!X1XXO.	2	8 (4,4)	3.20	
107	CB 6B	BIT 5,E	X!X1XXO.	2	8 (4,4)	3.20	
108	CB 6C	BIT 5,H	X!X1XXO.	2	8 (4,4)	3.20	
109	CB 6D	BIT 5,L	X!X1XXO.	2	8 (4,4)	3.20	
110	CB 76	BIT 6,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
111	DD CB 05 76	BIT 6,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
112	FD CB 05 76	BIT 6,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
113	CB 77	BIT 6,A	X!X1XXO.	2	8 (4,4)	3.20	
114	CB 70	BIT 6,B	X!X1XXO.	2	8 (4,4)	3.20	
115	CB 71	BIT 6,C	X!X1XXO.	2	8 (4,4)	3.20	
116	CB 72	BIT 6,D	X!X1XXO.	2	8 (4,4)	3.20	
117	CB 73	BIT 6,E	X!X1XXO.	2	8 (4,4)	3.20	
118	CB 74	BIT 6,H	X!X1XXO.	2	8 (4,4)	3.20	
119	CB 75	BIT 6,L	X!X1XXO.	2	8 (4,4)	3.20	
120	CB 7E	BIT 7,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
121	DD CB 05 7E	BIT 7,(IX+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	
122	FD CB 05 7E	BIT 7,(IY+IND)	X!X1XXO.	5	20 (4,4,3,5,4)	8.00	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
123	CB 7F	BIT 7,A	X!X1XXO.	2	8 (4,4)	3.20	
124	CB 78	BIT 7,B	X!X1XXO.	2	8 (4,4)	3.20	
125	CB 79	BIT 7,C	X!X1XXO.	2	8 (4,4)	3.20	
126	CB 7A	BIT 7,D	X!X1XXO.	2	8 (4,4)	3.20	
127	CB 7B	BIT 7,E	X!X1XXO.	2	8 (4,4)	3.20	
128	CB 7C	BIT 7,H	X!X1XXO.	2	8 (4,4)	3.20	
129	CB 7D	BIT 7,L	X!X1XXO.	2	8 (4,4)	3.20	
130	DC BB AA	CALL C,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA CY=1
				3	10 (4,3,3)	4.00	DACA CY=0
131	FC BB AA	CALL M,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA S =1
				3	10 (4,3,3)	4.00	DACA S =0
132	D4 BB AA	CALL NC,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA CY=0
				3	10 (4,3,3)	4.00	DACA CY=1
133	CD BB AA	CALL NN	..X.X...	5	17 (4,3,4,3,3)	6.80	
134	C4 BB AA	CALL NZ,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA Z =0
				3	10 (4,3,3)	4.00	DACA Z =1
135	F4 BB AA	CALL P,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA S =0
				3	10 (4,3,3)	4.00	DACA S =1
136	EC BB AA	CALL PE,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA P =1
				3	10 (4,3,3)	4.00	DACA P =0
137	E4 BB AA	CALL PO,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA P =0
				3	10 (4,3,3)	4.00	DACA P =1
138	CC BB AA	CALL Z,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA Z =1
				3	10 (4,3,3)	4.00	DACA Z =0
139	3F	CCF	..XXX.O!	1	4 (4)	1.60	
140	BE	CP (HL)	!!X!XV1!	2	7 (4,3)	2.80	
141	DD BE 05	CP (IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
142	FD BE 05	CP (IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
143	BF	CP A	!!X!XV1!	1	4 (4)	1.60	
144	B8	CP B	!!X!XV1!	1	4 (4)	1.60	
145	B9	CP C	!!X!XV1!	1	4 (4)	1.60	
146	BA	CP D	!!X!XV1!	1	4 (4)	1.60	
147	BB	CP E	!!X!XV1!	1	4 (4)	1.60	
148	BC	CP H	!!X!XV1!	1	4 (4)	1.60	
149	BD	CP L	!!X!XV1!	1	4 (4)	1.60	
150	FE 20	CP N	!!X!XV1!	2	7 (4,3)	2.80	
151	ED A9	CPD	!!X!X!1.	4	16 (4,4,3,5)	6.40	
152	ED B9	CPDR	!!X!X!1.	5	21 (4,4,3,5,5)	8.40	DACA BC#O SI A#(HL)
				4	16 (4,4,3,5)	6.40	DACA BC=O SAU A=(HL)
153	ED A1	CPI	!!X!X!1.	4	16 (4,4,3,5)	6.40	
154	ED B1	CPIR	!!X!X!1.	5	21 (4,4,3,5,5)	8.40	DACA BC#O SI A#(HL)
				4	16 (4,4,3,5)	6.40	DACA BC=O SAU A=(HL)
155	2F	CPL	..X1X.1.	1	4 (4)	1.60	
156	27	DAA	!!X!XP.!	1	4 (4)	1.60	
157	35	DEC (HL)	!!X!XV1.	3	11 (4,4,3)	4.40	
158	DD 35 05	DEC (IX+IND)	!!X!XV1.	6	23 (4,4,3,5,4,3)	9.20	
159	FD 35 05	DEC (IY+IND)	!!X!XV1.	6	23 (4,4,3,5,4,3)	9.20	
160	3D	DEC A	!!X!XV1.	1	4 (4)	1.60	
161	05	DEC B	!!X!XV1.	1	4 (4)	1.60	
162	0B	DEC BC	..X.X...	1	6 (6)	2.40	
163	0D	DEC C	!!X!XV1.	1	4 (4)	1.60	
164	15	DEC D	!!X!XV1.	1	4 (4)	1.60	
165	1B	DEC DE	..X.X...	1	6 (6)	2.40	
166	1D	DEC E	!!X!XV1.	1	4 (4)	1.60	
167	25	DEC H	!!X!XV1.	1	4 (4)	1.60	
168	2B	DEC HL	..X.X...	1	6 (6)	2.40	
169	DD 2B	DEC IX	..X.X...	2	10 (4,6)	4.00	
170	FD 2B	DEC IY	..X.X...	2	10 (4,6)	4.00	
171	2D	DEC L	!!X!XV1.	1	4 (4)	1.60	
172	3B	DEC SP	..X.X...	1	6 (6)	2.40	
173	F3	DI	..X.X...	1	4 (4)	1.60	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
174	10 03	DJNZ $+DIS	..X.X...	3	13 (5,3,5)	5.20	DACA B # 0
				2	8 (5,3)	3.20	DACA B = 0
175	FB	EI	..X.X...	1	4 (4)	1.60	
176	E3	EX (SP),HL	..X.X...	5	19 (4,3,4,3,5)	7.60	
177	DD E3	EX (SP),IX	..X.X...	6	23 (4,4,3,4,3,5)	9.20	
178	FD E3	EX (SP),IY	..X.X...	6	23 (4,4,3,4,3,5)	9.20	
179	08	EX AF,AF'	!!!!!!!!	1	4 (4)	1.60	
180	EB	EX DE,HL	..X.X...	1	4 (4)	1.60	
181	D9	EXX	..X.X...	1	4 (4)	1.60	
182	76	HALT	..X.X...	1	4 (4)	1.60	
183	ED 46	IM 0	..X.X...	2	8 (4,4)	3.20	
184	ED 56	IM 1	..X.X...	2	8 (4,4)	3.20	
185	ED 5E	IM 2	..X.X...	2	8 (4,4)	3.20	
186	ED 78	IN A,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
187	DB 20	IN A,(N)	..X.X...	3	11 (4,3,4)	4.40	
188	ED 40	IN B,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
189	ED 48	IN C,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
190	ED 50	IN D,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
191	ED 58	IN E,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
192	ED 60	IN H,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
193	ED 68	IN L,(C)	!!X!XPO.	3	12 (4,4,4)	4.80	
194	34	INC (HL)	!!X!XVO.	3	11 (4,4,3)	4.40	
195	DD 34 05	INC (IX+IND)	!!X!XVO.	6	23 (4,4,3,5,4,3)	9.20	
196	FD 34 05	INC (IY+IND)	!!X!XVO.	6	23 (4,4,3,5,4,3)	9.20	
197	3C	INC A	!!X!XVO.	1	4 (4)	1.60	
198	04	INC B	!!X!XVO.	1	4 (4)	1.60	
199	03	INC BC	..X.X...	1	6 (6)	2.40	
200	0C	INC C	!!X!XVO.	1	4 (4)	1.60	
201	14	INC D	!!X!XVO.	1	4 (4)	1.60	
202	13	INC DE	..X.X...	1	6 (6)	2.40	
203	1C	INC E	!!X!XVO.	1	4 (4)	1.60	
204	24	INC H	!!X!XVO.	1	4 (4)	1.60	
205	23	INC HL	..X.X...	1	6 (6)	2.40	
206	DD 23	INC IX	..X.X...	2	10 (4,6)	4.00	
207	FD 23	INC IY	..X.X...	2	10 (4,6)	4.00	
208	2C	INC L	!!X!XVO.	1	4 (4)	1.60	
209	33	INC SP	..X.X...	1	6 (6)	2.40	
210	ED AA	IND	X!XXXX1.	4	16 (4,5,4,3)	6.40	
211	ED BA	INDR	X1XXXX1.	5	21 (4,5,4,3,5)	8.40	DACA B # 0
				4	16 (4,5,4,3)	6.40	DACA B = 0
212	ED A2	INI	X!XXXX1.	4	16 (4,5,4,3)	6.40	
213	ED B2	INIR	X1XXXX1.	5	21 (4,5,4,3,5)	8.40	DACA B # 0
				4	16 (4,5,4,3)	6.40	DACA B = 0
214	E9	JP (HL)	..X.X...	1	4 (4)	1.60	
215	DD E9	JP (IX)	..X.X...	2	8 (4,4)	3.20	
216	FD E9	JP (IY)	..X.X...	2	8 (4,4)	3.20	
217	DA BB AA	JP C,NN	..X.X...	3	10 (4,3,3)	4.00	
218	FA BB AA	JP M,NN	..X.X...	3	10 (4,3,3)	4.00	
219	D2 BB AA	JP NC,NN	..X.X...	3	10 (4,3,3)	4.00	
220	C3 BB AA	JP NN	..X.X...	3	10 (4,3,3)	4.00	
221	C2 BB AA	JP NZ,NN	..X.X...	3	10 (4,3,3)	4.00	
222	F2 BB AA	JP P,NN	..X.X...	3	10 (4,3,3)	4.00	
223	EA BB AA	JP PE,NN	..X.X...	3	10 (4,3,3)	4.00	
224	E2 BB AA	JP PO,NN	..X.X...	3	10 (4,3,3)	4.00	
225	CA BB AA	JP Z,NN	..X.X...	3	10 (4,3,3)	4.00	
226	38 03	JR C,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA CY=1
				2	7 (4,3)	2.80	DACA CY=0
227	18 03	JR $+DIS	..X.X...	3	12 (4,3,5)	4.80	
228	30 03	JR NC,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA CY=0
				2	7 (4,3)	2.80	DACA CY=1
229	20 03	JR NZ,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA Z =0

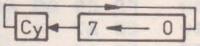
nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
				2	7 (4,3)	2.80	DACA Z =1
230	28 03	JR Z,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA Z =1
				2	7 (4,3)	2.80	DACA Z =0
231	02	LD (BC),A	..X.X...	2	7 (4,3)	2.80	
232	12	LD (DE),A	..X.X...	2	7 (4,3)	2.80	
233	77	LD (HL),A	..X.X...	2	7 (4,3)	2.80	
234	70	LD (HL),B	..X.X...	2	7 (4,3)	2.80	
235	71	LD (HL),C	..X.X...	2	7 (4,3)	2.80	
236	72	LD (HL),D	..X.X...	2	7 (4,3)	2.80	
237	73	LD (HL),E	..X.X...	2	7 (4,3)	2.80	
238	74	LD (HL),H	..X.X...	2	7 (4,3)	2.80	
239	75	LD (HL),L	..X.X...	2	7 (4,3)	2.80	
240	36 20	LD (HL),N	..X.X...	3	10 (4,3,3)	4.00	
241	DD 77 05	LD (IX+IND),A	..X.X...	5	19 (4,4,3,5,3)	7.60	
242	DD 70 05	LD (IX+IND),B	..X.X...	5	19 (4,4,3,5,3)	7.60	
243	DD 71 05	LD (IX+IND),C	..X.X...	5	19 (4,4,3,5,3)	7.60	
244	DD 72 05	LD (IX+IND),D	..X.X...	5	19 (4,4,3,5,3)	7.60	
245	DD 73 05	LD (IX+IND),E	..X.X...	5	19 (4,4,3,5,3)	7.60	
246	DD 74 05	LD (IX+IND),H	..X.X...	5	19 (4,4,3,5,3)	7.60	
247	DD 75 05	LD (IX+IND),L	..X.X...	5	19 (4,4,3,5,3)	7.60	
248	DD 36 05 20	LD (IX+IND),N	..X.X...	5	19 (4,4,3,5,3)	7.60	
249	FD 77 05	LD (IY+IND),A	..X.X...	5	19 (4,4,3,5,3)	7.60	
250	FD 70 05	LD (IY+IND),B	..X.X...	5	19 (4,4,3,5,3)	7.60	
251	FD 71 05	LD (IY+IND),C	..X.X...	5	19 (4,4,3,5,3)	7.60	
252	FD 72 05	LD (IY+IND),D	..X.X...	5	19 (4,4,3,5,3)	7.60	
253	FD 73 05	LD (IY+IND),E	..X.X...	5	19 (4,4,3,5,3)	7.60	
254	FD 74 05	LD (IY+IND),H	..X.X...	5	19 (4,4,3,5,3)	7.60	
255	FD 75 05	LD (IY+IND),L	..X.X...	5	19 (4,4,3,5,3)	7.60	
256	FD 36 05 20	LD (IY+IND),N	..X.X...	5	19 (4,4,3,5,3)	7.60	
257	32 BB AA	LD (NN),A	..X.X...	4	13 (4,3,3,3)	5.20	
258	ED 43 BB AA	LD (NN),BC	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
259	ED 53 BB AA	LD (NN),DE	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
260	22 BB AA	LD (NN),HL	..X.X...	5	16 (4,3,3,3,3)	6.40	
261	ED 63 BB AA	LD (NN),HL	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
262	DD 22 BB AA	LD (NN),IX	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
263	FD 22 BB AA	LD (NN),IY	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
264	ED 73 BB AA	LD (NN),SP	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
265	0A	LD A,(BC)	..X.X...	2	7 (4,3)	2.80	
266	1A	LD A,(DE)	..X.X...	2	7 (4,3)	2.80	
267	7E	LD A,(HL)	..X.X...	2	7 (4,3)	2.80	
268	DD 7E 05	LD A,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
269	FD 7E 05	LD A,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
270	3A BB AA	LD A,(NN)	..X.X...	4	13 (4,3,3,3)	5.20	
271	7F	LD A,A	..X.X...	1	4 (4)	1.60	
272	78	LD A,B	..X.X...	1	4 (4)	1.60	
273	79	LD A,C	..X.X...	1	4 (4)	1.60	
274	7A	LD A,D	..X.X...	1	4 (4)	1.60	
275	7B	LD A,E	..X.X...	1	4 (4)	1.60	
276	7C	LD A,H	..X.X...	1	4 (4)	1.60	
277	7D	LD A,L	..X.X...	1	4 (4)	1.60	
278	ED 57	LD A,I	!!XOXIO.	2	9 (4,5)	3.60	
279	3E 20	LD A,N	..X.X...	2	7 (4,3)	2.80	
280	ED 5F	LD A,R	!!XOXIO.	2	9 (4,5)	3.60	
281	46	LD B,(HL)	..X.X...	2	7 (4,3)	2.80	
282	DD 46 05	LD B,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
283	FD 46 05	LD B,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
284	47	LD B,A	..X.X...	1	4 (4)	1.60	
285	40	LD B,B	..X.X...	1	4 (4)	1.60	
286	41	LD B,C	..X.X...	1	4 (4)	1.60	
287	42	LD B,D	..X.X...	1	4 (4)	1.60	
288	43	LD B,E	..X.X...	1	4 (4)	1.60	

nr. crt	cod	mnemonica		flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
289	44	LD	B,H	..X.X...	1	4 (4)	1.60	
290	45	LD	B,L	..X.X...	1	4 (4)	1.60	
291	06 20	LD	B,N	..X.X...	2	7 (4,3)	2.80	
292	ED 4B BB AA	LD	BC,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
293	01 BB AA	LD	BC,NN	..X.X...	3	10 (4,3,3)	4.00	
294	4E	LD	C,(HL)	..X.X...	2	7 (4,3)	2.80	
295	DD 4E 05	LD	C,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
296	FD 4E 05	LD	C,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
297	4F	LD	C,A	..X.X...	1	4 (4)	1.60	
298	48	LD	C,B	..X.X...	1	4 (4)	1.60	
299	49	LD	C,C	..X.X...	1	4 (4)	1.60	
300	4A	LD	C,D	..X.X...	1	4 (4)	1.60	
301	4B	LD	C,E	..X.X...	1	4 (4)	1.60	
302	4C	LD	C,H	..X.X...	1	4 (4)	1.60	
303	4D	LD	C,L	..X.X...	1	4 (4)	1.60	
304	0E 20	LD	C,N	..X.X...	2	7 (4,3)	2.80	
305	56	LD	D,(HL)	..X.X...	2	7 (4,3)	2.80	
306	DD 56 05	LD	D,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
307	FD 56 05	LD	D,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
308	57	LD	D,A	..X.X...	1	4 (4)	1.60	
309	50	LD	D,B	..X.X...	1	4 (4)	1.60	
310	51	LD	D,C	..X.X...	1	4 (4)	1.60	
311	52	LD	D,D	..X.X...	1	4 (4)	1.60	
312	53	LD	D,E	..X.X...	1	4 (4)	1.60	
313	54	LD	D,H	..X.X...	1	4 (4)	1.60	
314	55	LD	D,L	..X.X...	1	4 (4)	1.60	
315	16 20	LD	D,N	..X.X...	2	7 (4,3)	2.80	
316	ED 5B BB AA	LD	DE,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
317	11 BB AA	LD	DE,NN	..X.X...	3	10 (4,3,3)	4.00	
318	5E	LD	E,(HL)	..X.X...	2	7 (4,3)	2.80	
319	DD 5E 05	LD	E,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
320	FD 5E 05	LD	E,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
321	5F	LD	E,A	..X.X...	1	4 (4)	1.60	
322	58	LD	E,B	..X.X...	1	4 (4)	1.60	
323	59	LD	E,C	..X.X...	1	4 (4)	1.60	
324	5A	LD	E,D	..X.X...	1	4 (4)	1.60	
325	5B	LD	E,E	..X.X...	1	4 (4)	1.60	
326	5C	LD	E,H	..X.X...	1	4 (4)	1.60	
327	5D	LD	E,L	..X.X...	1	4 (4)	1.60	
328	1E 20	LD	E,N	..X.X...	2	7 (4,3)	2.80	
329	66	LD	H,(HL)	..X.X...	2	7 (4,3)	2.80	
330	DD 66 05	LD	H,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
331	FD 66 05	LD	H,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
332	67	LD	H,A	..X.X...	1	4 (4)	1.60	
333	60	LD	H,B	..X.X...	1	4 (4)	1.60	
334	61	LD	H,C	..X.X...	1	4 (4)	1.60	
335	62	LD	H,D	..X.X...	1	4 (4)	1.60	
336	63	LD	H,E	..X.X...	1	4 (4)	1.60	
337	64	LD	H,H	..X.X...	1	4 (4)	1.60	
338	65	LD	H,L	..X.X...	1	4 (4)	1.60	
339	26 20	LD	H,N	..X.X...	2	7 (4,3)	2.80	
340	2A BB AA	LD	HL,(NN)	..X.X...	5	16 (4,3,3,3,3)	6.40	
341	ED 6B BB AA	LD	HL,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
342	21 BB AA	LD	HL,NN	..X.X...	3	10 (4,3,3)	4.00	
343	ED 47	LD	I,A	..X.X...	2	9 (4,5)	3.60	
344	DD 2A BB AA	LD	IX,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
345	DD 21 BB AA	LD	IX,NN	..X.X...	4	14 (4,4,3,3)	5.60	
346	FD 2A BB AA	LD	IY,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
347	FD 21 BB AA	LD	IY,NN	..X.X...	4	14 (4,4,3,3)	5.60	
348	6E	LD	L,(HL)	..X.X...	2	7 (4,3)	2.80	
349	DD 6E 05	LD	L,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	

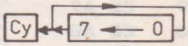
nr. crt	cod	mnemonica		flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
350	FD 6E 05	LD	L,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
351	6F	LD	L,A	..X.X...	1	4 (4)	1.60	
352	68	LD	L,B	..X.X...	1	4 (4)	1.60	
353	69	LD	L,C	..X.X...	1	4 (4)	1.60	
354	6A	LD	L,D	..X.X...	1	4 (4)	1.60	
355	6B	LD	L,E	..X.X...	1	4 (4)	1.60	
356	6C	LD	L,H	..X.X...	1	4 (4)	1.60	
357	6D	LD	L,L	..X.X...	1	4 (4)	1.60	
358	2E 20	LD	L,N	..X.X...	2	7 (4,3)	2.80	
359	ED 4F	LD	R,A	..X.X...	2	9 (4,5)	3.60	
360	ED 7B BB AA	LD	SP,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
361	F9	LD	SP,HL	..X.X...	1	6 (6)	2.40	
362	DD F9	LD	SP,IX	..X.X...	2	10 (4,6)	4.00	
363	FD F9	LD	SP,IY	..X.X...	2	10 (4,6)	4.00	
364	31 BB AA	LD	SP,NN	..X.X...	3	10 (4,3,3)	4.00	
365	ED A8	LDD		..X0X!0.	4	16 (4,4,3,5)	6.40	
366	ED B8	LDDR		..X0X00.	5	21 (4,4,3,5,5)	8.40	DACA BC # 0
					4	16 (4,4,3,5)	6.40	DACA BC = 0
367	ED A0	LDI		..X0X!0.	4	16 (4,4,3,5)	6.40	
368	ED B0	LDIR		..X0X00.	5	21 (4,4,3,5,5)	8.40	DACA BC # 0
					4	16 (4,4,3,5)	6.40	DACA BC = 0
369	ED 44	NEG		!!X!XV!!	2	8 (4,4)	3.20	
370	00	NOP		..X.X...	1	4 (4)	1.60	
371	B6	OR	(HL)	!!X0XP00	2	7 (4,3)	2.80	
372	DD B6 05	OR	(IX+IND)	!!X0XP00	5	19 (4,4,3,5,3)	7.60	
373	FD B6 05	OR	(IY+IND)	!!X0XP00	5	19 (4,4,3,5,3)	7.60	
374	B7	OR	A	!!X0XP00	1	4 (4)	1.60	
375	B0	OR	B	!!X0XP00	1	4 (4)	1.60	
376	B1	OR	C	!!X0XP00	1	4 (4)	1.60	
377	B2	OR	D	!!X0XP00	1	4 (4)	1.60	
378	B3	OR	E	!!X0XP00	1	4 (4)	1.60	
379	B4	OR	H	!!X0XP00	1	4 (4)	1.60	
380	B5	OR	L	!!X0XP00	1	4 (4)	1.60	
381	F6 20	OR	N	!!X0XP00	2	7 (4,3)	2.80	
382	ED BB	OTDR		X1XXXX1.	5	21 (4,5,3,4,5)	8.40	DACA B # 0
					4	16 (4,5,3,4)	6.40	DACA B = 0
383	ED B3	OTIR		X1XXXX1.	5	21 (4,5,3,4,5)	8.40	DACA B # 0
					4	16 (4,5,3,4)	6.40	DACA B = 0
384	ED 79	OUT	(C),A	..X.X...	3	12 (4,4,4)	4.80	
385	ED 41	OUT	(C),B	..X.X...	3	12 (4,4,4)	4.80	
386	ED 49	OUT	(C),C	..X.X...	3	12 (4,4,4)	4.80	
387	ED 51	OUT	(C),D	..X.X...	3	12 (4,4,4)	4.80	
388	ED 59	OUT	(C),E	..X.X...	3	12 (4,4,4)	4.80	
389	ED 61	OUT	(C),H	..X.X...	3	12 (4,4,4)	4.80	
390	ED 69	OUT	(C),L	..X.X...	3	12 (4,4,4)	4.80	
391	D3 20	OUT	(N),A	..X.X...	3	11 (4,3,4)	4.40	
392	ED AB	OUTD		X1XXXX1.	4	16 (4,5,3,4)	6.40	
393	ED A3	OUTI		X1XXXX1.	4	16 (4,5,3,4)	6.40	
394	F1	POP	AF	!!!!!!!!	3	10 (4,3,3)	4.00	
395	C1	POP	BC	..X.X...	3	10 (4,3,3)	4.00	
396	D1	POP	DE	..X.X...	3	10 (4,3,3)	4.00	
397	E1	POP	HL	..X.X...	3	10 (4,3,3)	4.00	
398	DD E1	POP	IX	..X.X...	4	14 (4,4,3,3)	5.60	
399	FD E1	POP	IY	..X.X...	4	14 (4,4,3,3)	5.60	
400	F5	PUSH	AF	..X.X...	3	11 (5,3,3)	4.40	
401	C5	PUSH	BC	..X.X...	3	11 (5,3,3)	4.40	
402	D5	PUSH	DE	..X.X...	3	11 (5,3,3)	4.40	
403	E5	PUSH	HL	..X.X...	3	11 (5,3,3)	4.40	
404	DD E5	PUSH	IX	..X.X...	4	15 (4,5,3,3)	6.00	
405	FD E5	PUSH	IY	..X.X...	4	15 (4,5,3,3)	6.00	
406	CB 86	RES	0,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
407	DD CB 05 86	RES 0,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
408	FD CB 05 86	RES 0,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
409	CB 87	RES 0,A	..X.X...	2	8 (4,4)	3.20	
410	CB 80	RES 0,B	..X.X...	2	8 (4,4)	3.20	
411	CB 81	RES 0,C	..X.X...	2	8 (4,4)	3.20	
412	CB 82	RES 0,D	..X.X...	2	8 (4,4)	3.20	
413	CB 83	RES 0,E	..X.X...	2	8 (4,4)	3.20	
414	CB 84	RES 0,H	..X.X...	2	8 (4,4)	3.20	
415	CB 85	RES 0,L	..X.X...	2	8 (4,4)	3.20	
416	CB 8E	RES 1,(HL)	..X.X...	4	15 (4,4,4,3)	6.20	
417	DD CB 05 8E	RES 1,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
418	FD CB 05 8E	RES 1,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
419	CB 8F	RES 1,A	..X.X...	2	8 (4,4)	3.20	
420	CB 88	RES 1,B	..X.X...	2	8 (4,4)	3.20	
421	CB 89	RES 1,C	..X.X...	2	8 (4,4)	3.20	
422	CB 8A	RES 1,D	..X.X...	2	8 (4,4)	3.20	
423	CB 8B	RES 1,E	..X.X...	2	8 (4,4)	3.20	
424	CB 8C	RES 1,H	..X.X...	2	8 (4,4)	3.20	
425	CB 8D	RES 1,L	..X.X...	2	8 (4,4)	3.20	
426	CB 96	RES 2,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
427	DD CB 05 96	RES 2,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
428	FD CB 05 96	RES 2,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
429	CB 97	RES 2,A	..X.X...	2	8 (4,4)	3.20	
430	CB 90	RES 2,B	..X.X...	2	8 (4,4)	3.20	
431	CB 91	RES 2,C	..X.X...	2	8 (4,4)	3.20	
432	CB 92	RES 2,D	..X.X...	2	8 (4,4)	3.20	
433	CB 93	RES 2,E	..X.X...	2	8 (4,4)	3.20	
434	CB 94	RES 2,H	..X.X...	2	8 (4,4)	3.20	
435	CB 95	RES 2,L	..X.X...	2	8 (4,4)	3.20	
436	CB 9E	RES 3,(HL)	..X.X...	3	15 (4,4,4,3)	6.00	
437	DD CB 05 9E	RES 3,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
438	FD CB 05 9E	RES 3,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
439	CB 9F	RES 3,A	..X.X...	2	8 (4,4)	3.20	
440	CB 98	RES 3,B	..X.X...	2	8 (4,4)	3.20	
441	CB 99	RES 3,C	..X.X...	2	8 (4,4)	3.20	
442	CB 9A	RES 3,D	..X.X...	2	8 (4,4)	3.20	
443	CB 9B	RES 3,E	..X.X...	2	8 (4,4)	3.20	
444	CB 9C	RES 3,H	..X.X...	2	8 (4,4)	3.20	
445	CB 9D	RES 3,L	..X.X...	2	8 (4,4)	3.20	
446	CB A6	RES 4,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
447	DD CB 05 A6	RES 4,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
448	FD CB 05 A6	RES 4,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
449	CB A7	RES 4,A	..X.X...	2	8 (4,4)	3.20	
450	CB A0	RES 4,B	..X.X...	2	8 (4,4)	3.20	
451	CB A1	RES 4,C	..X.X...	2	8 (4,4)	3.20	
452	CB A2	RES 4,D	..X.X...	2	8 (4,4)	3.20	
453	CB A3	RES 4,E	..X.X...	2	8 (4,4)	3.20	
454	CB A4	RES 4,H	..X.X...	2	8 (4,4)	3.20	
455	CB A5	RES 4,L	..X.X...	2	8 (4,4)	3.20	
456	CB AE	RES 5,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
457	DD CB 05 AE	RES 5,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
458	FD CB 05 AE	RES 5,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
459	CB AF	RES 5,A	..X.X...	2	8 (4,4)	3.20	
460	CB A8	RES 5,B	..X.X...	2	8 (4,4)	3.20	
461	CB A9	RES 5,C	..X.X...	2	8 (4,4)	3.20	
462	CB AA	RES 5,D	..X.X...	2	8 (4,4)	3.20	
463	CB AB	RES 5,E	..X.X...	2	8 (4,4)	3.20	
464	CB AC	RES 5,H	..X.X...	2	8 (4,4)	3.20	
465	CB AD	RES 5,L	..X.X...	2	8 (4,4)	3.20	
466	CB B6	RES 6,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
467	DD CB 05 B6	RES 6,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	

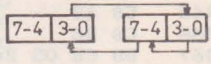
nr. crt	cod	mnemonica		flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
468	FD CB 05 B6	RES	6,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
469	CB B7	RES	6,A	..X.X...	2	8 (4,4)	3.20	
470	CB B0	RES	6,B	..X.X...	2	8 (4,4)	3.20	
471	CB B1	RES	6,C	..X.X...	2	8 (4,4)	3.20	
472	CB B2	RES	6,D	..X.X...	2	8 (4,4)	3.20	
473	CB B3	RES	6,E	..X.X...	2	8 (4,4)	3.20	
474	CB B4	RES	6,H	..X.X...	2	8 (4,4)	3.20	
475	CB B5	RES	6,L	..X.X...	2	8 (4,4)	3.20	
476	CB BE	RES	7,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
477	DD CB 05 BE	RES	7,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
478	FD CB 05 BE	RES	7,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
479	CB BF	RES	7,A	..X.X...	2	8 (4,4)	3.20	
480	CB B8	RES	7,B	..X.X...	2	8 (4,4)	3.20	
481	CB B9	RES	7,C	..X.X...	2	8 (4,4)	3.20	
482	CB BA	RES	7,D	..X.X...	2	8 (4,4)	3.20	
483	CB BB	RES	7,E	..X.X...	2	8 (4,4)	3.20	
484	CB BC	RES	7,H	..X.X...	2	8 (4,4)	3.20	
485	CB BD	RES	7,L	..X.X...	2	8 (4,4)	3.20	
486	C9	RET		..X.X...	3	10 (4,3,3)	4.00	
487	D8	RET	C	..X.X...	3	11 (5,3,3)	4.40	DACA CY=1
					1	5 (5)	2.00	DACA CY=0
488	F8	RET	M	..X.X...	3	11 (5,3,3)	4.40	DACA S =1
					5	5 (5)	2.00	DACA S =0
489	D0	RET	NC	..X.X...	3	11 (5,3,3)	4.40	DACA CY=0
					1	5 (5)	2.00	DACA CY=1
490	C0	RET	NZ	..X.X...	3	11 (5,3,3)	4.40	DACA Z =0
					1	5 (5)	2.00	DACA Z =1
491	F0	RET	P	..X.X...	3	11 (5,3,3)	4.40	DACA S =0
					1	5 (5)	2.00	DACA S =1
492	E8	RET	PE	..X.X...	3	11 (5,3,3)	4.40	DACA P =1
					1	5 (5)	2.00	DACA P =0
493	E0	RET	PO	..X.X...	3	11 (5,3,3)	4.40	DACA P =0
					1	5 (5)	2.00	DACA P =1
494	C8	RET	Z	..X.X...	3	11 (5,3,3)	4.40	DACA Z =1
					1	5 (5)	2.00	DACA Z =0
495	ED 4D	RETI		..X.X...	4	14 (4,4,3,3)	5.60	
496	ED 45	RETN		..X.X...	4	14 (4,4,3,3)	5.60	
497	CB 16	RL	(HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
498	DD CB 05 16	RL	(IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
499	FD CB 05 16	RL	(IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
500	CB 17	RL	A	!!XOXPO!	2	8 (4,4)	3.20	
501	CB 10	RL	B	!!XOXPO!	2	8 (4,4)	3.20	
502	CB 11	RL	C	!!XOXPO!	2	8 (4,4)	3.20	
503	CB 12	RL	D	!!XOXPO!	2	8 (4,4)	3.20	
504	CB 13	RL	E	!!XOXPO!	2	8 (4,4)	3.20	
505	CB 14	RL	H	!!XOXPO!	2	8 (4,4)	3.20	
506	CB 15	RL	L	!!XOXPO!	2	8 (4,4)	3.20	
507	17	RLA		..XOX.O!	1	4 (4)	1.60	
508	CB 06	RLC	(HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
509	DD CB 05 06	RLC	(IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
510	FD CB 05 06	RLC	(IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
511	CB 07	RLC	A	!!XOXPO!	2	8 (4,4)	3.20	
512	CB 00	RLC	B	!!XOXPO!	2	8 (4,4)	3.20	
513	CB 01	RLC	C	!!XOXPO!	2	8 (4,4)	3.20	
514	CB 02	RLC	D	!!XOXPO!	2	8 (4,4)	3.20	
515	CB 03	RLC	E	!!XOXPO!	2	8 (4,4)	3.20	
516	CB 04	RLC	H	!!XOXPO!	2	8 (4,4)	3.20	
517	CB 05	RLC	L	!!XOXPO!	2	8 (4,4)	3.20	
518	07	RLCA		..XOX.O!	1	4 (4)	1.60	
519	ED 6F	RLD		!!XOXPO.	5	18 (4,4,3,4,3)	7.20	
520	CB 1E	RR	(HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	



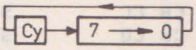
byte

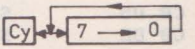


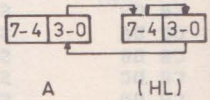
byte



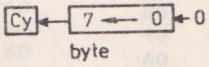
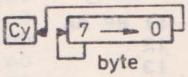
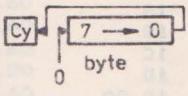
nr. crt	cod	mnemonica		flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
521	DD CB 05 1E	RR	(IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
522	FD CB 05 1E	RR	(IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
523	CB 1F	RR	A	!!XOXPO!	2	8 (4,4)	3.20	
524	CB 18	RR	B	!!XOXPO!	2	8 (4,4)	3.20	
525	CB 19	RR	C	!!XOXPO!	2	8 (4,4)	3.20	
526	CB 1A	RR	D	!!XOXPO!	2	8 (4,4)	3.20	
527	CB 1B	RR	E	!!XOXPO!	2	8 (4,4)	3.20	
528	CB 1C	RR	H	!!XOXPO!	2	8 (4,4)	3.20	
529	CB 1D	RR	L	!!XOXPO!	2	8 (4,4)	3.20	
530	1F	RRA		..XOX.0!	1	4 (4)	1.60	
531	CB 0E	RRC	(HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
532	DD CB 05 0E	RRC	(IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
533	FD CB 05 0E	RRC	(IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
534	CB 0F	RRC	A	!!XOXPO!	2	8 (4,4)	3.20	
535	CB 08	RRC	B	!!XOXPO!	2	8 (4,4)	3.20	
536	CB 09	RRC	C	!!XOXPO!	2	8 (4,4)	3.20	
537	CB 0A	RRC	D	!!XOXPO!	2	8 (4,4)	3.20	
538	CB 0B	RRC	E	!!XOXPO!	2	8 (4,4)	3.20	
539	CB 0C	RRC	H	!!XOXPO!	2	8 (4,4)	3.20	
540	CB 0D	RRC	L	!!XOXPO!	2	8 (4,4)	3.20	
541	0F	RRCA		..XOX.0!	1	4 (4)	1.60	
542	ED 67	RRD		!!XOXPO.	5	18 (4,4,3,4,3)	7.20	
543	C7	RST	00H	..X.X...	3	11 (5,3,3)	4.40	
544	CF	RST	08H	..X.X...	3	11 (5,3,3)	4.40	
545	D7	RST	10H	..X.X...	3	11 (5,3,3)	4 40	
546	DF	RST	18H	..X.X...	3	11 (5,3,3)	4.40	
547	E7	RST	20H	..X.X...	3	11 (5,3,3)	4.40	
548	EF	RST	28H	..X.X...	3	11 (5,3,3)	4.40	
549	F7	RST	30H	..X.X...	3	11 (5,3,3)	4.40	
550	FF	RST	38H	..X.X...	3	11 (5,3,3)	4.40	
551	9E	SBC	A,(HL)	!!X!XV1!	2	7 (4,3)	2.80	
552	DD 9E 05	SBC	A,(IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
553	FD 9E 05	SBC	A,(IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
554	9F	SBC	A,A	!!X!XV1!	1	4 (4)	1.60	
555	98	SBC	A,B	!!X!XV1!	1	4 (4)	1.60	
556	99	SBC	A,C	!!X!XV1!	1	4 (4)	1.60	
557	9A	SBC	A,D	!!X!XV1!	1	4 (4)	1.60	
558	9B	SBC	A,E	!!X!XV1!	1	4 (4)	1.60	
559	9C	SBC	A,H	!!X!XV1!	1	4 (4)	1.60	
560	9D	SBC	A,L	!!X!XV1!	1	4 (4)	1.60	
561	DE 20	SBC	A,N	!!X!XV1!	2	7 (4,3)	2.80	
562	ED 42	SBC	HL,BC	!!X!XV1!	4	15 (4,4,4,3)	6.00	
563	ED 52	SBC	HL,DE	!!X!XV1!	4	15 (4,4,4,3)	6.00	
564	ED 62	SBC	HL,HL	!!X!XV1!	4	15 (4,4,4,3)	6.00	
565	ED 72	SBC	HL,SP	!!X!XV1!	4	15 (4,4,4,3)	6.00	
566	37	SCF		..XOX.01	1	4 (4)	1.60	
567	CB C6	SET	0,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
568	DD CB 05 C6	SET	0,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
569	FD CB 05 C6	SET	0,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
570	CB C7	SET	0,A	..X.X...	2	8 (4,4)	3.20	
571	CB C0	SET	0,B	..X.X...	2	8 (4,4)	3.20	
572	CB C1	SET	0,C	..X.X...	2	8 (4,4)	3.20	
573	CB C2	SET	0,D	..X.X...	2	8 (4,4)	3.20	
574	CB C3	SET	0,E	..X.X...	2	8 (4,4)	3.20	
575	CB C4	SET	0,H	..X.X...	2	8 (4,4)	3.20	
576	CB C5	SET	0,L	..X.X...	2	8 (4,4)	3.20	
577	CB CE	SET	1,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
578	DD CB 05 CE	SET	1,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
579	FD CB 05 CE	SET	1,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
580	CB CF	SET	1,A	..X.X...	2	8 (4,4)	3.20	
581	CB C8	SET	1,B	..X.X...	2	8 (4,4)	3.20	







nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
582	CB C9	SET 1,C	..X.X...	2	8 (4,4)	3.20	
583	CB CA	SET 1,D	..X.X...	2	8 (4,4)	3.20	
584	CB CB	SET 1,E	..X.X...	2	8 (4,4)	3.20	
585	CB CC	SET 1,H	..X.X...	2	8 (4,4)	3.20	
586	CB CD	SET 1,L	..X.X...	2	8 (4,4)	3.20	
587	CB D6	SET 2,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
588	DD CB 05 D6	SET 2,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
589	FD CB 05 D6	SET 2,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
590	CB D7	SET 2,A	..X.X...	2	8 (4,4)	3.20	
591	CB D0	SET 2,B	..X.X...	2	8 (4,4)	3.20	
592	CB D1	SET 2,C	..X.X...	2	8 (4,4)	3.20	
593	CB D2	SET 2,D	..X.X...	2	8 (4,4)	3.20	
594	CB D3	SET 2,E	..X.X...	2	8 (4,4)	3.20	
595	CB D4	SET 2,H	..X.X...	2	8 (4,4)	3.20	
596	CB D5	SET 2,L	..X.X...	2	8 (4,4)	3.20	
597	CB DE	SET 3,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
598	DD CB 05 DE	SET 3,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
599	FD CB 05 DE	SET 3,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
600	CB DF	SET 3,A	..X.X...	2	8 (4,4)	3.20	
601	CB D8	SET 3,B	..X.X...	2	8 (4,4)	3.20	
602	CB D9	SET 3,C	..X.X...	2	8 (4,4)	3.20	
603	CB DA	SET 3,D	..X.X...	2	8 (4,4)	3.20	
604	CB DB	SET 3,E	..X.X...	2	8 (4,4)	3.20	
605	CB DC	SET 3,H	..X.X...	2	8 (4,4)	3.20	
606	CB DD	SET 3,L	..X.X...	2	8 (4,4)	3.20	
607	CB E6	SET 4,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
608	DD CB 05 E6	SET 4,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
609	FD CB 05 E6	SET 4,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
610	CB E7	SET 4,A	..X.X...	2	8 (4,4)	3.20	
611	CB E0	SET 4,B	..X.X...	2	8 (4,4)	3.20	
612	CB E1	SET 4,C	..X.X...	2	8 (4,4)	3.20	
613	CB E2	SET 4,D	..X.X...	2	8 (4,4)	3.20	
614	CB E3	SET 4,E	..X.X...	2	8 (4,4)	3.20	
615	CB E4	SET 4,H	..X.X...	2	8 (4,4)	3.20	
616	CB E5	SET 4,L	..X.X...	2	8 (4,4)	3.20	
617	CB EE	SET 5,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
618	DD CB 05 EE	SET 5,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
619	FD CB 05 EE	SET 5,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
620	CB EF	SET 5,A	..X.X...	2	8 (4,4)	3.20	
621	CB E8	SET 5,B	..X.X...	2	8 (4,4)	3.20	
622	CB E9	SET 5,C	..X.X...	2	8 (4,4)	3.20	
623	CB EA	SET 5,D	..X.X...	2	8 (4,4)	3.20	
624	CB EB	SET 5,E	..X.X...	2	8 (4,4)	3.20	
625	CB EC	SET 5,H	..X.X...	2	8 (4,4)	3.20	
626	CB ED	SET 5,L	..X.X...	2	8 (4,4)	3.20	
627	CB F6	SET 6,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
628	DD CB 05 F6	SET 6,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
629	FD CB 05 F6	SET 6,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
630	CB F7	SET 6,A	..X.X...	2	8 (4,4)	3.20	
631	CB F0	SET 6,B	..X.X...	2	8 (4,4)	3.20	
632	CB F1	SET 6,C	..X.X...	2	8 (4,4)	3.20	
633	CB F2	SET 6,D	..X.X...	2	8 (4,4)	3.20	
634	CB F3	SET 6,E	..X.X...	2	8 (4,4)	3.20	
635	CB F4	SET 6,H	..X.X...	2	8 (4,4)	3.20	
636	CB F5	SET 6,L	..X.X...	2	8 (4,4)	3.20	
637	CB FE	SET 7,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
638	DD CB 05 FE	SET 7,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
639	FD CB 05 FE	SET 7,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
640	CB FF	SET 7,A	..X.X...	2	8 (4,4)	3.20	
641	CB F8	SET 7,B	..X.X...	2	8 (4,4)	3.20	
642	CB F9	SET 7,C	..X.X...	2	8 (4,4)	3.20	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
643	CB FA	SET 7,D	..X.X...	2	8 (4,4)	3.20	
644	CB FB	SET 7,E	..X.X...	2	8 (4,4)	3.20	
645	CB FC	SET 7,H	..X.X...	2	8 (4,4)	3.20	
646	CB FD	SET 7,L	..X.X...	2	8 (4,4)	3.20	
647	CB 26	SLA (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
648	DD CB 05 26	SLA (IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
649	FD CB 05 26	SLA (IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
650	CB 27	SLA A	!!XOXPO!	2	8 (4,4)	3.20	
651	CB 20	SLA B	!!XOXPO!	2	8 (4,4)	3.20	
652	CB 21	SLA C	!!XOXPO!	2	8 (4,4)	3.20	
653	CB 22	SLA D	!!XOXPO!	2	8 (4,4)	3.20	
654	CB 23	SLA E	!!XOXPO!	2	8 (4,4)	3.20	
655	CB 24	SLA H	!!XOXPO!	2	8 (4,4)	3.20	
656	CB 25	SLA L	!!XOXPO!	2	8 (4,4)	3.20	
657	CB 2E	SRA (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
658	DD CB 05 2E	SRA (IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
659	FD CB 05 2E	SRA (IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
660	CB 2F	SRA A	!!XOXPO!	2	8 (4,4)	3.20	
661	CB 28	SRA B	!!XOXPO!	2	8 (4,4)	3.20	
662	CB 29	SRA C	!!XOXPO!	2	8 (4,4)	3.20	
663	CB 2A	SRA D	!!XOXPO!	2	8 (4,4)	3.20	
664	CB 2B	SRA E	!!XOXPO!	2	8 (4,4)	3.20	
665	CB 2C	SRA H	!!XOXPO!	2	8 (4,4)	3.20	
666	CB 2D	SRA L	!!XOXPO!	2	8 (4,4)	3.20	
667	CB 3E	SRL (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
668	DD CB 05 3E	SRL (IX+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
669	FD CB 05 3E	SRL (IY+IND)	!!XOXPO!	6	23 (4,4,3,5,4,3)	9.20	
670	CB 3F	SRL A	!!XOXPO!	2	8 (4,4)	3.20	
671	CB 38	SRL B	!!XOXPO!	2	8 (4,4)	3.20	
672	CB 39	SRL C	!!XOXPO!	2	8 (4,4)	3.20	
673	CB 3A	SRL D	!!XOXPO!	2	8 (4,4)	3.20	
674	CB 3B	SRL E	!!XOXPO!	2	8 (4,4)	3.20	
675	CB 3C	SRL H	!!XOXPO!	2	8 (4,4)	3.20	
676	CB 3D	SRL L	!!XOXPO!	2	8 (4,4)	3.20	
677	96	SUB (HL)	!!X!XV1!	2	7 (4,3)	2.80	
678	DD 96 05	SUB (IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
679	FD 96 05	SUB (IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
680	97	SUB A	!!X!XV1!	1	4 (4)	1.60	
681	90	SUB B	!!X!XV1!	1	4 (4)	1.60	
682	91	SUB C	!!X!XV1!	1	4 (4)	1.60	
683	92	SUB D	!!X!XV1!	1	4 (4)	1.60	
684	93	SUB E	!!X!XV1!	1	4 (4)	1.60	
685	94	SUB H	!!X!XV1!	1	4 (4)	1.60	
686	95	SUB L	!!X!XV1!	1	4 (4)	1.60	
687	D6 20	SUB N	!!X!XV1!	2	7 (4,3)	2.80	
688	AE	XOR (HL)	!!XOXP00	2	7 (4,3)	2.80	
689	DD AE 05	XOR (IX+IND)	!!XOXP00	5	19 (4,4,3,5,3)	7.60	
690	FD AE 05	XOR (IY+IND)	!!XOXP00	5	19 (4,4,3,5,3)	7.60	
691	AF	XOR A	!!XOXP00	1	4 (4)	1.60	
692	A8	XOR B	!!XOXP00	1	4 (4)	1.60	
693	A9	XOR C	!!XOXP00	1	4 (4)	1.60	
694	AA	XOR D	!!XOXP00	1	4 (4)	1.60	
695	AB	XOR E	!!XOXP00	1	4 (4)	1.60	
696	AC	XOR H	!!XOXP00	1	4 (4)	1.60	
697	AD	XOR L	!!XOXP00	1	4 (4)	1.60	
698	EE 20	XOR N	!!XOXP00	2	7 (4,3)	2.80	

L-C-1

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
1	00	NOP	..X.X...	1	4 (4)	1.60	
2	01 BB AA	LD BC,NN	..X.X...	3	10 (4,3,3)	4.00	
3	02	LD (BC),A	..X.X...	2	7 (4,3)	2.80	
4	03	INC BC	..X.X...	1	6 (6)	2.40	
5	04	INC B	!!X!XVO.	1	4 (4)	1.60	
6	05	DEC B	!!X!XV1.	1	4 (4)	1.60	
7	06 20	LD B,N	..X.X...	2	7 (4,3)	2.80	
8	07	RLCA	..XOX.0!	1	4 (4)	1.60	
9	08	EX AF,AF'	!!!!!!!!	1	4 (4)	1.60	
10	09	ADD HL,BC	..X!X.0!	3	11 (4,4,3)	4.40	
11	0A	LD A,(BC)	..X.X...	2	7 (4,3)	2.80	
12	0B	DEC BC	..X.X...	1	6 (6)	2.40	
13	0C	INC C	!!X!XVO.	1	4 (4)	1.60	
14	0D	DEC C	!!X!XV1.	1	4 (4)	1.60	
15	0E 20	LD C,N	..X.X...	2	7 (4,3)	2.80	
16	0F	RRCA	..XOX.0!	1	4 (4)	1.60	
17	10 03	DJNZ $+DIS	..X.X...	3	13 (5,3,5)	5.20	DACA B # 0
				2	8 (5,3)	3.20	DACA B = 0
18	11 BB AA	LD DE,NN	..X.X...	3	10 (4,3,3)	4.00	
19	12	LD (DE),A	..X.X...	2	7 (4,3)	2.80	
20	13	INC DE	..X.X...	1	6 (6)	2.40	
21	14	INC D	!!X!XVO.	1	4 (4)	1.60	
22	15	DEC D	!!X!XV1.	1	4 (4)	1.60	
23	16 20	LD D,N	..X.X...	2	7 (4,3)	2.80	
24	17	RLA	..XOX.0!	1	4 (4)	1.60	
25	18 03	JR $+DIS	..X.X...	3	12 (4,3,5)	4.80	
26	19	ADD HL,DE	..X!X.0!	3	11 (4,4,3)	4.40	
27	1A	LD A,(DE)	..X.X...	2	7 (4,3)	2.80	
28	1B	DEC DE	..X.X...	1	6 (6)	2.40	
29	1C	INC E	!!X!XVO.	1	4 (4)	1.60	
30	1D	DEC E	!!X!XV1.	1	4 (4)	1.60	
31	1E 20	LD E,N	..X.X...	2	7 (4,3)	2.80	
32	1F	RRA	..XOX.0!	1	4 (4)	1.60	
33	20 03	JR NZ,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA Z =0
				2	7 (4,3)	2.80	DACA Z =1
34	21 BB AA	LD HL,NN	..X.X...	3	10 (4,3,3)	4.00	
35	22 BB AA	LD (NN),HL	..X.X...	5	16 (4,3,3,3,3)	6.40	
36	23	INC HL	..X.X...	1	6 (6)	2.40	
37	24	INC H	!!X!XVO.	1	4 (4)	1.60	
38	25	DEC H	!!X!XV1.	1	4 (4)	1.60	
39	26 20	LD H,N	..X.X...	2	7 (4,3)	2.80	
40	27	DAA	!!X!XP.!	1	4 (4)	1.60	
41	28 03	JR Z,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA Z =1
				2	7 (4,3)	2.80	DACA Z =0
42	29	ADD HL,HL	..X!X.0!	3	11 (4,4,3)	4.40	
43	2A BB AA	LD HL,(NN)	..X.X...	5	16 (4,3,3,3,3)	6.40	
44	2B	DEC HL	..X.X...	1	6 (6)	2.40	
45	2C	INC L	!!X!XVO.	1	4 (4)	1.60	
46	2D	DEC L	!!X!XV1.	1	4 (4)	1.60	
47	2E 20	LD L,N	..X.X...	2	7 (4,3)	2.80	
48	2F	CPL	..X1X.1.	1	4 (4)	1.60	
49	30 03	JR NC,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA CY=0
				2	7 (4,3)	2.80	DACA CY=1
50	31 BB AA	LD SP,NN	..X.X...	3	10 (4,3,3)	4.00	
51	32 BB AA	LD (NN),A	..X.X...	4	13 (4,3,3,3)	5.20	
52	33	INC SP	..X.X...	1	6 (6)	2.40	
53	34	INC (HL)	!!X!XVO.	3	11 (4,4,3)	4.40	
54	35	DEC (HL)	!!X!XV1.	3	11 (4,4,3)	4.40	
55	36 20	LD (HL),N	..X.X...	3	10 (4,3,3)	4.00	
56	37	SCF	..XOX.01	1	4 (4)	1.60	
57	38 03	JR C,$+DIS	..X.X...	3	12 (4,3,5)	4.80	DACA CY=1

nr. crt	cod	mnemonica		flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
					2	7 (4,3)	2.80	DACA CY=0
58	39	ADD	HL,SP	..X!X.0!	3	11 (4,4,3)	4.40	
59	3A BB AA	LD	A,(NN)	..X.X...	4	13 (4,3,3,3)	5.20	
60	3B	DEC	SP	..X.X...	1	6 (6)	2.40	
61	3C	INC	A	!!X!XV0.	1	4 (4)	1.60	
62	3D	DEC	A	!!X!XV1.	1	4 (4)	1.60	
63	3E 20	LD	A,N	..X.X...	2	7 (4,3)	2.80	
64	3F	CCF		..XXX.0!	1	4 (4)	1.60	
65	40	LD	B,B	..X.X...	1	4 (4)	1.60	
66	41	LD	B,C	..X.X...	1	4 (4)	1.60	
67	42	LD	B,D	..X.X...	1	4 (4)	1.60	
68	43	LD	B,E	..X.X...	1	4 (4)	1.60	
69	44	LD	B,H	..X.X...	1	4 (4)	1.60	
70	45	LD	B,L	..X.X...	1	4 (4)	1.60	
71	46	LD	B,(HL)	..X.X...	2	7 (4,3)	2.80	
72	47	LD	B,A	..X.X...	1	4 (4)	1.60	
73	48	LD	C,B	..X.X...	1	4 (4)	1.60	
74	49	LD	C,C	..X.X...	1	4 (4)	1.60	
75	4A	LD	C,D	..X.X...	1	4 (4)	1.60	
76	4B	LD	C,E	..X.X...	1	4 (4)	1.60	
77	4C	LD	C,H	..X.X...	1	4 (4)	1.60	
78	4D	LD	C,L	..X.X...	1	4 (4)	1.60	
79	4E	LD	C,(HL)	..X.X...	2	7 (4,3)	2.80	
80	4F	LD	C,A	..X.X...	1	4 (4)	1.60	
81	50	LD	D,B	..X.X...	1	4 (4)	1.60	
82	51	LD	D,C	..X.X...	1	4 (4)	1.60	
83	52	LD	D,D	..X.X...	1	4 (4)	1.60	
84	53	LD	D,E	..X.X...	1	4 (4)	1.60	
85	54	LD	D,H	..X.X...	1	4 (4)	1.60	
86	55	LD	D,L	..X.X...	1	4 (4)	1.60	
87	56	LD	D,(HL)	..X.X...	2	7 (4,3)	2.80	
88	57	LD	D,A	..X.X...	1	4 (4)	1.60	
89	58	LD	E,B	..X.X...	1	4 (4)	1.60	
90	59	LD	E,C	..X.X...	1	4 (4)	1.60	
91	5A	LD	E,D	..X.X...	1	4 (4)	1.60	
92	5B	LD	E,E	..X.X...	1	4 (4)	1.60	
93	5C	LD	E,H	..X.X...	1	4 (4)	1.60	
94	5D	LD	E,L	..X.X...	1	4 (4)	1.60	
95	5E	LD	E,(HL)	..X.X...	2	7 (4,3)	2.80	
96	5F	LD	E,A	..X.X...	1	4 (4)	1.60	
97	60	LD	H,B	..X.X...	1	4 (4)	1.60	
98	61	LD	H,C	..X.X...	1	4 (4)	1.60	
99	62	LD	H,D	..X.X...	1	4 (4)	1.60	
100	63	LD	H,E	..X.X...	1	4 (4)	1.60	
101	64	LD	H,H	..X.X...	1	4 (4)	1.60	
102	65	LD	H,L	..X.X...	1	4 (4)	1.60	
103	66	LD	H,(HL)	..X.X...	2	7 (4,3)	2.80	
104	67	LD	H,A	..X.X...	1	4 (4)	1.60	
105	68	LD	L,B	..X.X...	1	4 (4)	1.60	
106	69	LD	L,C	..X.X...	1	4 (4)	1.60	
107	6A	LD	L,D	..X.X...	1	4 (4)	1.60	
108	6B	LD	L,E	..X.X...	1	4 (4)	1.60	
109	6C	LD	L,H	..X.X...	1	4 (4)	1.60	
110	6D	LD	L,L	..X.X...	1	4 (4)	1.60	
111	6E	LD	L,(HL)	..X.X...	2	7 (4,3)	2.80	
112	6F	LD	L,A	..X.X...	1	4 (4)	1.60	
113	70	LD	(HL),B	..X.X...	2	7 (4,3)	2.80	
114	71	LD	(HL),C	..X.X...	2	7 (4,3)	2.80	
115	72	LD	(HL),D	..X.X...	2	7 (4,3)	2.80	
116	73	LD	(HL),E	..X.X...	2	7 (4,3)	2.80	
117	74	LD	(HL),H	..X.X...	2	7 (4,3)	2.80	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
118	75	LD (HL),L	..X.X...	2	7 (4,3)	2.80	
119	76	HALT	..X.X...	1	4 (4)	1.60	
120	77	LD (HL),A	..X.X...	2	7 (4,3)	2.80	
121	78	LD A,B	..X.X...	1	4 (4)	1.60	
122	79	LD A,C	..X.X...	1	4 (4)	1.60	
123	7A	LD A,D	..X.X...	1	4 (4)	1.60	
124	7B	LD A,E	..X.X...	1	4 (4)	1.60	
125	7C	LD A,H	..X.X...	1	4 (4)	1.60	
126	7D	LD A,L	..X.X...	1	4 (4)	1.60	
127	7E	LD A,(HL)	..X.X...	2	7 (4,3)	2.80	
128	7F	LD A,A	..X.X...	1	4 (4)	1.60	
129	80	ADD A,B	!!X!XVO!	1	4 (4)	1.60	
130	81	ADD A,C	!!X!XVO!	1	4 (4)	1.60	
131	82	ADD A,D	!!X!XVO!	1	4 (4)	1.60	
132	83	ADD A,E	!!X!XVO!	1	4 (4)	1.60	
133	84	ADD A,H	!!X!XVO!	1	4 (4)	1.60	
134	85	ADD A,L	!!X!XVO!	1	4 (4)	1.60	
135	86	ADD A,(HL)	!!X!XVO!	2	7 (4,3)	2.80	
136	87	ADD A,A	!!X!XVO!	1	4 (4)	1.60	
137	88	ADC A,B	!!X!XVO!	1	4 (4)	1.60	
138	89	ADC A,C	!!X!XVO!	1	4 (4)	1.60	
139	8A	ADC A,D	!!X!XVO!	1	4 (4)	1.60	
140	8B	ADC A,E	!!X!XVO!	1	4 (4)	1.60	
141	8C	ADC A,H	!!X!XVO!	1	4 (4)	1.60	
142	8D	ADC A,L	!!X!XVO!	1	4 (4)	1.60	
143	8E	ADC A,(HL)	!!X!XVO!	2	7 (4,3)	2.80	
144	8F	ADC A,A	!!X!XVO!	1	4 (4)	1.60	
145	90	SUB B	!!X!XV1!	1	4 (4)	1.60	
146	91	SUB C	!!X!XV1!	1	4 (4)	1.60	
147	92	SUB D	!!X!XV1!	1	4 (4)	1.60	
148	93	SUB E	!!X!XV1!	1	4 (4)	1.60	
149	94	SUB H	!!X!XV1!	1	4 (4)	1.60	
150	95	SUB L	!!X!XV1!	1	4 (4)	1.60	
151	96	SUB (HL)	!!X!XV1!	2	7 (4,3)	2.80	
152	97	SUB A	!!X!XV1!	1	4 (4)	1.60	
153	98	SBC A,B	!!X!XV1!	1	4 (4)	1.60	
154	99	SBC A,C	!!X!XV1!	1	4 (4)	1.60	
155	9A	SBC A,D	!!X!XV1!	1	4 (4)	1.60	
156	9B	SBC A,E	!!X!XV1!	1	4 (4)	1.60	
157	9C	SBC A,H	!!X!XV1!	1	4 (4)	1.60	
158	9D	SBC A,L	!!X!XV1!	1	4 (4)	1.60	
159	9E	SBC A,(HL)	!!X!XV1!	2	7 (4,3)	2.80	
160	9F	SBC A,A	!!X!XV1!	1	4 (4)	1.60	
161	A0	AND B	!!X1XPOO	1	4 (4)	1.60	
162	A1	AND C	!!X1XPOO	1	4 (4)	1.60	
163	A2	AND D	!!X1XPOO	1	4 (4)	1.60	
164	A3	AND E	!!X1XPOO	1	4 (4)	1.60	
165	A4	AND H	!!X1XPOO	1	4 (4)	1.60	
166	A5	AND L	!!X1XPOO	1	4 (4)	1.60	
167	A6	AND (HL)	!!X1XPOO	2	7 (4,3)	2.80	
168	A7	AND A	!!X1XPOO	1	4 (4)	1.60	
169	A8	XOR B	!!XOXPOO	1	4 (4)	1.60	
170	A9	XOR C	!!XOXPOO	1	4 (4)	1.60	
171	AA	XOR D	!!XOXPOO	1	4 (4)	1.60	
172	AB	XOR E	!!XOXPOO	1	4 (4)	1.60	
173	AC	XOR H	!!XOXPOO	1	4 (4)	1.60	
174	AD	XOR L	!!XOXPOO	1	4 (4)	1.60	
175	AE	XOR (HL)	!!XOXPOO	2	7 (4,3)	2.80	
176	AF	XOR A	!!XOXPOO	1	4 (4)	1.60	
177	B0	OR B	!!XOXPOO	1	4 (4)	1.60	
178	B1	OR C	!!XOXPOO	1	4 (4)	1.60	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
179	B2	OR D	!!XOXPOO	1	4 (4)	1.60	
180	B3	OR E	!!XOXPOO	1	4 (4)	1.60	
181	B4	OR H	!!XOXPOO	1	4 (4)	1.60	
182	B5	OR L	!!XOXPOO	1	4 (4)	1.60	
183	B6	OR (HL)	!!XOXPOO	2	7 (4,3)	2.80	
184	B7	OR A	!!XOXPOO	1	4 (4)	1.60	
185	B8	CP B	!!X!XV1!	1	4 (4)	1.60	
186	B9	CP C	!!X!XV1!	1	4 (4)	1.60	
187	BA	CP D	!!X!XV1!	1	4 (4)	1.60	
188	BB	CP E	!!X!XV1!	1	4 (4)	1.60	
189	BC	CP H	!!X!XV1!	1	4 (4)	1.60	
190	BD	CP L	!!X!XV1!	1	4 (4)	1.60	
191	BE	CP (HL)	!!X!XV1!	2	7 (4,3)	2.80	
192	BF	CP A	!!X!XV1!	1	4 (4)	1.60	
193	C0	RET NZ	..X.X...	3	11 (5,3,3)	4.40	DACA Z =0
				1	5 (5)	2.00	DACA Z =1
194	C1	POP BC	..X.X...	3	10 (4,3,3)	4.00	
195	C2 BB AA	JP NZ,NN	..X.X...	3	10 (4,3,3)	4.00	
196	C3 BB AA	JP NN	..X.X...	3	10 (4,3,3)	4.00	
197	C4 BB AA	CALL NZ,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA Z =0
				3	10 (4,3,3)	4.00	DACA Z =1
198	C5	PUSH BC	..X.X...	3	11 (5,3,3)	4.40	
199	C6 20	ADD A,N	!!X!XVO!	2	7 (4,3)	2.80	
200	C7	RST OOH	..X.X...	3	11 (5,3,3)	4.40	
201	C8	RET Z	..X.X...	3	11 (5,3,3)	4.40	DACA Z =1
				1	5 (5)	2.00	DACA Z =0
202	C9	RET	..X.X...	3	10 (4,3,3)	4.00	
203	CA BB AA	JP Z,NN	..X.X...	3	10 (4,3,3)	4.00	
204	CB 00	RLC B	!!XOXPO!	2	8 (4,4)	3.20	
205	CB 01	RLC C	!!XOXPO!	2	8 (4,4)	3.20	
206	CB 02	RLC D	!!XOXPO!	2	8 (4,4)	3.20	
207	CB 03	RLC E	!!XOXPO!	2	8 (4,4)	3.20	
208	CB 04	RLC H	!!XOXPO!	2	8 (4,4)	3.20	
209	CB 05	RLC L	!!XOXPO!	2	8 (4,4)	3.20	
210	CB 06	RLC (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
211	CB 07	RLC A	!!XOXPO!	2	8 (4,4)	3.20	
212	CB 08	RRC B	!!XOXPO!	2	8 (4,4)	3.20	
213	CB 09	RRC C	!!XOXPO!	2	8 (4,4)	3.20	
214	CB 0A	RRC D	!!XOXPO!	2	8 (4,4)	3.20	
215	CB 0B	RRC E	!!XOXPO!	2	8 (4,4)	3.20	
216	CB 0C	RRC H	!!XOXPO!	2	8 (4,4)	3.20	
217	CB 0D	RRC L	!!XOXPO!	2	8 (4,4)	3.20	
218	CB 0E	RRC (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
219	CB 0F	RRC A	!!XOXPO!	2	8 (4,4)	3.20	
220	CB 10	RL B	!!XOXPO!	2	8 (4,4)	3.20	
221	CB 11	RL C	!!XOXPO!	2	8 (4,4)	3.20	
222	CB 12	RL D	!!XOXPO!	2	8 (4,4)	3.20	
223	CB 13	RL E	!!XOXPO!	2	8 (4,4)	3.20	
224	CB 14	RL H	!!XOXPO!	2	8 (4,4)	3.20	
225	CB 15	RL L	!!XOXPO!	2	8 (4,4)	3.20	
226	CB 16	RL (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
227	CB 17	RL A	!!XOXPO!	2	8 (4,4)	3.20	
228	CB 18	RR B	!!XOXPO!	2	8 (4,4)	3.20	
229	CB 19	RR C	!!XOXPO!	2	8 (4,4)	3.20	
230	CB 1A	RR D	!!XOXPO!	2	8 (4,4)	3.20	
231	CB 1B	RR E	!!XOXPO!	2	8 (4,4)	3.20	
232	CB 1C	RR H	!!XOXPO!	2	8 (4,4)	3.20	
233	CB 1D	RR L	!!XOXPO!	2	8 (4,4)	3.20	
234	CB 1E	RR (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
235	CB 1F	RR A	!!XOXPO!	2	8 (4,4)	3.20	
236	CB 20	SLA B	!!XOXPO!	2	8 (4,4)	3.20	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
237	CB 21	SLA C	!!XOXPO!	2	8 (4,4)	3.20	
238	CB 22	SLA D	!!XOXPO!	2	8 (4,4)	3.20	
239	CB 23	SLA E	!!XOXPO!	2	8 (4,4)	3.20	
240	CB 24	SLA H	!!XOXPO!	2	8 (4,4)	3.20	
241	CB 25	SLA L	!!XOXPO!	2	8 (4,4)	3.20	
242	CB 26	SLA (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
243	CB 27	SLA A	!!XOXPO!	2	8 (4,4)	3.20	
244	CB 28	SRA B	!!XOXPO!	2	8 (4,4)	3.20	
245	CB 29	SRA C	!!XOXPO!	2	8 (4,4)	3.20	
246	CB 2A	SRA D	!!XOXPO!	2	8 (4,4)	3.20	
247	CB 2B	SRA E	!!XOXPO!	2	8 (4,4)	3.20	
248	CB 2C	SRA H	!!XOXPO!	2	8 (4,4)	3.20	
249	CB 2D	SRA L	!!XOXPO!	2	8 (4,4)	3.20	
250	CB 2E	SRA (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
251	CB 2F	SRA A	!!XOXPO!	2	8 (4,4)	3.20	
252	CB 38	SRL B	!!XOXPO!	2	8 (4,4)	3.20	
253	CB 39	SRL C	!!XOXPO!	2	8 (4,4)	3.20	
254	CB 3A	SRL D	!!XOXPO!	2	8 (4,4)	3.20	
255	CB 3B	SRL E	!!XOXPO!	2	8 (4,4)	3.20	
256	CB 3C	SRL H	!!XOXPO!	2	8 (4,4)	3.20	
257	CB 3D	SRL L	!!XOXPO!	2	8 (4,4)	3.20	
258	CB 3E	SRL (HL)	!!XOXPO!	4	15 (4,4,4,3)	6.00	
259	CB 3F	SRL A	!!XOXPO!	2	8 (4,4)	3.20	
260	CB 40	BIT 0,B	X!X1XXO.	2	8 (4,4)	3.20	
261	CB 41	BIT 0,C	X!X1XXO.	2	8 (4,4)	3.20	
262	CB 42	BIT 0,D	X!X1XXO.	2	8 (4,4)	3.20	
263	CB 43	BIT 0,E	X!X1XXO.	2	8 (4,4)	3.20	
264	CB 44	BIT 0,H	X!X1XXO.	2	8 (4,4)	3.20	
265	CB 45	BIT 0,L	X!X1XXO.	2	8 (4,4)	3.20	
266	CB 46	BIT 0,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
267	CB 47	BIT 0,A	X!X1XXO.	2	8 (4,4)	3.20	
268	CB 48	BIT 1,B	X!X1XXO.	2	8 (4,4)	3.20	
269	CB 49	BIT 1,C	X!X1XXO.	2	8 (4,4)	3.20	
270	CB 4A	BIT 1,D	X!X1XXO.	2	8 (4,4)	3.20	
271	CB 4B	BIT 1,E	X!X1XXO.	2	8 (4,4)	3.20	
272	CB 4C	BIT 1,H	X!X1XXO.	2	8 (4,4)	3.20	
273	CB 4D	BIT 1,L	X!X1XXO.	2	8 (4,4)	3.20	
274	CB 4E	BIT 1,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
275	CB 4F	BIT 1,A	X!X1XXO.	2	8 (4,4)	3.20	
276	CB 50	BIT 2,B	X!X1XXO.	2	8 (4,4)	3.20	
277	CB 51	BIT 2,C	X!X1XXO.	2	8 (4,4)	3.20	
278	CB 52	BIT 2,D	X!X1XXO.	2	8 (4,4)	3.20	
279	CB 53	BIT 2,E	X!X1XXO.	2	8 (4,4)	3.20	
280	CB 54	BIT 2,H	X!X1XXO.	2	8 (4,4)	3.20	
281	CB 55	BIT 2,L	X!X1XXO.	2	8 (4,4)	3.20	
282	CB 56	BIT 2,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
283	CB 57	BIT 2,A	X!X1XXO.	2	8 (4,4)	3.20	
284	CB 58	BIT 3,B	X!X1XXO.	2	8 (4,4)	3.20	
285	CB 59	BIT 3,C	X!X1XXO.	2	8 (4,4)	3.20	
286	CB 5A	BIT 3,D	X!X1XXO.	2	8 (4,4)	3.20	
287	CB 5B	BIT 3,E	X!X1XXO.	2	8 (4,4)	3.20	
288	CB 5C	BIT 3,H	X!X1XXO.	2	8 (4,4)	3.20	
289	CB 5D	BIT 3,L	X!X1XXO.	2	8 (4,4)	3.20	
290	CB 5E	BIT 3,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
291	CB 5F	BIT 3,A	X!X1XXO.	2	8 (4,4)	3.20	
292	CB 60	BIT 4,B	X!X1XXO.	2	8 (4,4)	3.20	
293	CB 61	BIT 4,C	X!X1XXO.	2	8 (4,4)	3.20	
294	CB 62	BIT 4,D	X!X1XXO.	2	8 (4,4)	3.20	
295	CB 63	BIT 4,E	X!X1XXO.	2	8 (4,4)	3.20	
296	CB 64	BIT 4,H	X!X1XXO.	2	8 (4,4)	3.20	
297	CB 65	BIT 4,L	X!X1XXO.	2	8 (4,4)	3.20	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
298	CB 66	BIT 4,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
299	CB 67	BIT 4,A	X!X1XXO.	2	8 (4,4)	3.20	
300	CB 68	BIT 5,B	X!X1XXO.	2	8 (4,4)	3.20	
301	CB 69	BIT 5,C	X!X1XXO.	2	8 (4,4)	3.20	
302	CB 6A	BIT 5,D	X!X1XXO.	2	8 (4,4)	3.20	
303	CB 6B	BIT 5,E	X!X1XXO.	2	8 (4,4)	3.20	
304	CB 6C	BIT 5,H	X!X1XXO.	2	8 (4,4)	3.20	
305	CB 6D	BIT 5,L	X!X1XXO.	2	8 (4,4)	3.20	
306	CB 6E	BIT 5,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
307	CB 6F	BIT 5,A	X!X1XXO.	2	8 (4,4)	3.20	
308	CB 70	BIT 6,B	X!X1XXO.	2	8 (4,4)	3.20	
309	CB 71	BIT 6,C	X!X1XXO.	2	8 (4,4)	3.20	
310	CB 72	BIT 6,D	X!X1XXO.	2	8 (4,4)	3.20	
311	CB 73	BIT 6,E	X!X1XXO.	2	8 (4,4)	3.20	
312	CB 74	BIT 6,H	X!X1XXO.	2	8 (4,4)	3.20	
313	CB 75	BIT 6,L	X!X1XXO.	2	8 (4,4)	3.20	
314	CB 76	BIT 6,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
315	CB 77	BIT 6,A	X!X1XXO.	2	8 (4,4)	3.20	
316	CB 78	BIT 7,B	X!X1XXO.	2	8 (4,4)	3.20	
317	CB 79	BIT 7,C	X!X1XXO.	2	8 (4,4)	3.20	
318	CB 7A	BIT 7,D	X!X1XXO.	2	8 (4,4)	3.20	
319	CB 7B	BIT 7,E	X!X1XXO.	2	8 (4,4)	3.20	
320	CB 7C	BIT 7,H	X!X1XXO.	2	8 (4,4)	3.20	
321	CB 7D	BIT 7,L	X!X1XXO.	2	8 (4,4)	3.20	
322	CB 7E	BIT 7,(HL)	X!X1XXO.	3	12 (4,4,4)	4.80	
323	CB 7F	BIT 7,A	X!X1XXO.	2	8 (4,4)	3.20	
324	CB 80	RES 0,B	..X.X...	2	8 (4,4)	3.20	
325	CB 81	RES 0,C	..X.X...	2	8 (4,4)	3.20	
326	CB 82	RES 0,D	..X.X...	2	8 (4,4)	3.20	
327	CB 83	RES 0,E	..X.X...	2	8 (4,4)	3.20	
328	CB 84	RES 0,H	..X.X...	2	8 (4,4)	3.20	
329	CB 85	RES 0,L	..X.X...	2	8 (4,4)	3.20	
330	CB 86	RES 0,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
331	CB 87	RES 0,A	..X.X...	2	8 (4,4)	3.20	
332	CB 88	RES 1,B	..X.X...	2	8 (4,4)	3.20	
333	CB 89	RES 1,C	..X.X...	2	8 (4,4)	3.20	
334	CB 8A	RES 1,D	..X.X...	2	8 (4,4)	3.20	
335	CB 8B	RES 1,E	..X.X...	2	8 (4,4)	3.20	
336	CB 8C	RES 1,H	..X.X...	2	8 (4,4)	3.20	
337	CB 8D	RES 1,L	..X.X...	2	8 (4,4)	3.20	
338	CB 8E	RES 1,(HL)	..X.X...	4	15 (4,4,4,3)	6.20	
339	CB 8F	RES 1,A	..X.X...	2	8 (4,4)	3.20	
340	CB 90	RES 2,B	..X.X...	2	8 (4,4)	3.20	
341	CB 91	RES 2,C	..X.X...	2	8 (4,4)	3.20	
342	CB 92	RES 2,D	..X.X...	2	8 (4,4)	3.20	
343	CB 93	RES 2,E	..X.X...	2	8 (4,4)	3.20	
344	CB 94	RES 2,H	..X.X...	2	8 (4,4)	3.20	
345	CB 95	RES 2,L	..X.X...	2	8 (4,4)	3.20	
346	CB 96	RES 2,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
347	CB 97	RES 2,A	..X.X...	2	8 (4,4)	3.20	
348	CB 98	RES 3,B	..X.X...	2	8 (4,4)	3.20	
349	CB 99	RES 3,C	..X.X...	2	8 (4,4)	3.20	
350	CB 9A	RES 3,D	..X.X...	2	8 (4,4)	3.20	
351	CB 9B	RES 3,E	..X.X...	2	8 (4,4)	3.20	
352	CB 9C	RES 3,H	..X.X...	2	8 (4,4)	3.20	
353	CB 9D	RES 3,L	..X.X...	2	8 (4,4)	3.20	
354	CB 9E	RES 3,(HL)	..X.X...	3	15 (4,4,4,3)	6.00	
355	CB 9F	RES 3,A	..X.X...	2	8 (4,4)	3.20	
356	CB A0	RES 4,B	..X.X...	2	8 (4,4)	3.20	
357	CB A1	RES 4,C	..X.X...	2	8 (4,4)	3.20	
358	CB A2	RES 4,D	..X.X...	2	8 (4,4)	3.20	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
359	CB A3	RES 4,E	..X.X...	2	8 (4,4)	3.20	
360	CB A4	RES 4,H	..X.X...	2	8 (4,4)	3.20	
361	CB A5	RES 4,L	..X.X...	2	8 (4,4)	3.20	
362	CB A6	RES 4,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
363	CB A7	RES 4,A	..X.X...	2	8 (4,4)	3.20	
364	CB A8	RES 5,B	..X.X...	2	8 (4,4)	3.20	
365	CB A9	RES 5,C	..X.X...	2	8 (4,4)	3.20	
366	CB AA	RES 5,D	..X.X...	2	8 (4,4)	3.20	
367	CB AB	RES 5,E	..X.X...	2	8 (4,4)	3.20	
368	CB AC	RES 5,H	..X.X...	2	8 (4,4)	3.20	
369	CB AD	RES 5,L	..X.X...	2	8 (4,4)	3.20	
370	CB AE	RES 5,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
371	CB AF	RES 5,A	..X.X...	2	8 (4,4)	3.20	
372	CB B0	RES 6,B	..X.X...	2	8 (4,4)	3.20	
373	CB B1	RES 6,C	..X.X...	2	8 (4,4)	3.20	
374	CB B2	RES 6,D	..X.X...	2	8 (4,4)	3.20	
375	CB B3	RES 6,E	..X.X...	2	8 (4,4)	3.20	
376	CB B4	RES 6,H	..X.X...	2	8 (4,4)	3.20	
377	CB B5	RES 6,L	..X.X...	2	8 (4,4)	3.20	
378	CB B6	RES 6,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
379	CB B7	RES 6,A	..X.X...	2	8 (4,4)	3.20	
380	CB B8	RES 7,B	..X.X...	2	8 (4,4)	3.20	
381	CB B9	RES 7,C	..X.X...	2	8 (4,4)	3.20	
382	CB BA	RES 7,D	..X.X...	2	8 (4,4)	3.20	
383	CB BB	RES 7,E	..X.X...	2	8 (4,4)	3.20	
384	CB BC	RES 7,H	..X.X...	2	8 (4,4)	3.20	
385	CB BD	RES 7,L	..X.X...	2	8 (4,4)	3.20	
386	CB BE	RES 7,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
387	CB BF	RES 7,A	..X.X...	2	8 (4,4)	3.20	
388	CB C0	SET 0,B	..X.X...	2	8 (4,4)	3.20	
389	CB C1	SET 0,C	..X.X...	2	8 (4,4)	3.20	
390	CB C2	SET 0,D	..X.X...	2	8 (4,4)	3.20	
391	CB C3	SET 0,E	..X.X...	2	8 (4,4)	3.20	
392	CB C4	SET 0,H	..X.X...	2	8 (4,4)	3.20	
393	CB C5	SET 0,L	..X.X...	2	8 (4,4)	3.20	
394	CB C6	SET 0,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
395	CB C7	SET 0,A	..X.X...	2	8 (4,4)	3.20	
396	CB C8	SET 1,B	..X.X...	2	8 (4,4)	3.20	
397	CB C9	SET 1,C	..X.X...	2	8 (4,4)	3.20	
398	CB CA	SET 1,D	..X.X...	2	8 (4,4)	3.20	
399	CB CB	SET 1,E	..X.X...	2	8 (4,4)	3.20	
400	CB CC	SET 1,H	..X.X...	2	8 (4,4)	3.20	
401	CB CD	SET 1,L	..X.X...	2	8 (4,4)	3.20	
402	CB CE	SET 1,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
403	CB CF	SET 1,A	..X.X...	2	8 (4,4)	3.20	
404	CB D0	SET 2,B	..X.X...	2	8 (4,4)	3.20	
405	CB D1	SET 2,C	..X.X...	2	8 (4,4)	3.20	
406	CB D2	SET 2,D	..X.X...	2	8 (4,4)	3.20	
407	CB D3	SET 2,E	..X.X...	2	8 (4,4)	3.20	
408	CB D4	SET 2,H	..X.X...	2	8 (4,4)	3.20	
409	CB D5	SET 2,L	..X.X...	2	8 (4,4)	3.20	
410	CB D6	SET 2,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
411	CB D7	SET 2,A	..X.X...	2	8 (4,4)	3.20	
412	CB D8	SET 3,B	..X.X...	2	8 (4,4)	3.20	
413	CB D9	SET 3,C	..X.X...	2	8 (4,4)	3.20	
414	CB DA	SET 3,D	..X.X...	2	8 (4,4)	3.20	
415	CB DB	SET 3,E	..X.X...	2	8 (4,4)	3.20	
416	CB DC	SET 3,H	..X.X...	2	8 (4,4)	3.20	
417	CB DD	SET 3,L	..X.X...	2	8 (4,4)	3.20	
418	CB DE	SET 3,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
419	CB DF	SET 3,A	..X.X...	2	8 (4,4)	3.20	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
420	CB E0	SET 4,B	..X.X...	2	8 (4,4)	3.20	
421	CB E1	SET 4,C	..X.X...	2	8 (4,4)	3.20	
422	CB E2	SET 4,D	..X.X...	2	8 (4,4)	3.20	
423	CB E3	SET 4,E	..X.X...	2	8 (4,4)	3.20	
424	CB E4	SET 4,H	..X.X...	2	8 (4,4)	3.20	
425	CB E5	SET 4,L	..X.X...	2	8 (4,4)	3.20	
426	CB E6	SET 4,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
427	CB E7	SET 4,A	..X.X...	2	8 (4,4)	3.20	
428	CB E8	SET 5,B	..X.X...	2	8 (4,4)	3.20	
429	CB E9	SET 5,C	..X.X...	2	8 (4,4)	3.20	
430	CB EA	SET 5,D	..X.X...	2	8 (4,4)	3.20	
431	CB EB	SET 5,E	..X.X...	2	8 (4,4)	3.20	
432	CB EC	SET 5,H	..X.X...	2	8 (4,4)	3.20	
433	CB ED	SET 5,L	..X.X...	2	8 (4,4)	3.20	
434	CB EE	SET 5,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
435	CB EF	SET 5,A	..X.X...	2	8 (4,4)	3.20	
436	CB FO	SET 6,B	..X.X...	2	8 (4,4)	3.20	
437	CB F1	SET 6,C	..X.X...	2	8 (4,4)	3.20	
438	CB F2	SET 6,D	..X.X...	2	8 (4,4)	3.20	
439	CB F3	SET 6,E	..X.X...	2	8 (4,4)	3.20	
440	CB F4	SET 6,H	..X.X...	2	8 (4,4)	3.20	
441	CB F5	SET 6,L	..X.X...	2	8 (4,4)	3.20	
442	CB F6	SET 6,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
443	CB F7	SET 6,A	..X.X...	2	8 (4,4)	3.20	
444	CB F8	SET 7,B	..X.X...	2	8 (4,4)	3.20	
445	CB F9	SET 7,C	..X.X...	2	8 (4,4)	3.20	
446	CB FA	SET 7,D	..X.X...	2	8 (4,4)	3.20	
447	CB FB	SET 7,E	..X.X...	2	8 (4,4)	3.20	
448	CB FC	SET 7,H	..X.X...	2	8 (4,4)	3.20	
449	CB FD	SET 7,L	..X.X...	2	8 (4,4)	3.20	
450	CB FE	SET 7,(HL)	..X.X...	4	15 (4,4,4,3)	6.00	
451	CB FF	SET 7,A	..X.X...	2	8 (4,4)	3.20	
452	CC BB AA	CALL Z,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA Z =1
				3	10 (4,3,3)	4.00	DACA Z =0
453	CD BB AA	CALL NN	..X.X...	5	17 (4,3,4,3,3)	6.80	
454	CE 20	ADC A,N	!!X!XVO!	2	7 (4,3)	2.80	
455	CF	RST 08H	..X.X...	3	11 (5,3,3)	4.40	
456	DO	RET NC	..X.X...	3	11 (5,3,3)	4.40	DACA CY=0
				1	5 (5)	2.00	DACA CY=1
457	D1	POP DE	..X.X...	3	10 (4,3,3)	4.00	
458	D2 BB AA	JP NC,NN	..X.X...	3	10 (4,3,3)	4.00	
459	D3 20	OUT (N),A	..X.X...	3	11 (4,3,4)	4.40	
460	D4 BB AA	CALL NC,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA CY=0
				3	10 (4,3,3)	4.00	DACA CY=1
461	D5	PUSH DE	..X.X...	3	11 (5,3,3)	4.40	
462	D6 20	SUB N	!!X!XV1!	2	7 (4,3)	2.80	
463	D7	RST 10H	..X.X...	3	11 (5,3,3)	4.40	
464	D8	RET C	..X.X...	3	11 (5,3,3)	4.40	DACA CY=1
				1	5 (5)	2.00	DACA CY=0
465	D9	EXX	..X.X...	1	4 (4)	1.60	
466	DA BB AA	JP C,NN	..X.X...	3	10 (4,3,3)	4.00	
467	DB 20	IN A,(N)	..X.X...	3	11 (4,3,4)	4.40	
468	DC BB AA	CALL C,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA CY=1
				3	10 (4,3,3)	4.00	DACA CY=0
469	DD 09	ADD IX,BC	..X!X.0!	4	15 (4,4,4,3)	6.00	
470	DD 19	ADD IX,DE	..X!X.0!	4	15 (4,4,4,3)	6.00	
471	DD 21 BB AA	LD IX,NN	..X.X...	4	14 (4,4,3,3)	5.60	
472	DD 22 BB AA	LD (NN),IX	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
473	DD 23	INC IX	..X.X...	2	10 (4,6)	4.00	
474	DD 29	ADD IX,IX	..X!X.0!	4	15 (4,4,4,3)	6.00	
475	DD 2A BB AA	LD IX,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
476	DD 2B	DEC IX	..X.X...	2	10 (4,6)	4.00	
477	DD 34 05	INC (IX+IND)	!!X!XVO.	6	23 (4,4,3,5,4,3)	9.20	
478	DD 35 05	DEC (IX+IND)	!!X!XV1.	6	23 (4,4,3,5,4,3)	9.20	
479	DD 36 05 20	LD (IX+IND),N	..X.X...	5	19 (4,4,3,5,3)	7.60	
480	DD 39	ADD IX,SP	..X!X.O!	4	15 (4,4,4,3)	6.00	
481	DD 46 05	LD B,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
482	DD 4E 05	LD C,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
483	DD 56 05	LD D,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
484	DD 5E 05	LD E,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
485	DD 66 05	LD H,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
486	DD 6E 05	LD L,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
487	DD 70 05	LD (IX+IND),B	..X.X...	5	19 (4,4,3,5,3)	7.60	
488	DD 71 05	LD (IX+IND),C	..X.X...	5	19 (4,4,3,5,3)	7.60	
489	DD 72 05	LD (IX+IND),D	..X.X...	5	19 (4,4,3,5,3)	7.60	
490	DD 73 05	LD (IX+IND),E	..X.X...	5	19 (4,4,3,5,3)	7.60	
491	DD 74 05	LD (IX+IND),H	..X.X...	5	19 (4,4,3,5,3)	7.60	
492	DD 75 05	LD (IX+IND),L	..X.X...	5	19 (4,4,3,5,3)	7.60	
493	DD 77 05	LD (IX+IND),A	..X.X...	5	19 (4,4,3,5,3)	7.60	
494	DD 7E 05	LD A,(IX+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
495	DD 86 05	ADD A,(IX+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
496	DD 8E 05	ADC A,(IX+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
497	DD 96 05	SUB (IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
498	DD 9E 05	SBC A,(IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
499	DD A6 05	AND (IX+IND)	!!X1XP00	5	19 (4,4,3,5,3)	7.60	
500	DD AE 05	XOR (IX+IND)	!!X0XP00	5	19 (4,4,3,5,3)	7.60	
501	DD B6 05	OR (IX+IND)	!!X0XP00	5	19 (4,4,3,5,3)	7.60	
502	DD BE 05	CP (IX+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
503	DD CB 05 06	RLC (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
504	DD CB 05 0E	RRC (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
505	DD CB 05 16	RL (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
506	DD CB 05 1E	RR (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
507	DD CB 05 26	SLA (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
508	DD CB 05 2E	SRA (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
509	DD CB 05 3E	SRL (IX+IND)	!!X0XP0!	6	23 (4,4,3,5,4,3)	9.20	
510	DD CB 05 46	BIT 0,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
511	DD CB 05 4E	BIT 1,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
512	DD CB 05 56	BIT 2,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
513	DD CB 05 5E	BIT 3,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
514	DD CB 05 66	BIT 4,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
515	DD CB 05 6E	BIT 5,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
516	DD CB 05 76	BIT 6,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
517	DD CB 05 7E	BIT 7,(IX+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
518	DD CB 05 86	RES 0,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
519	DD CB 05 8E	RES 1,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
520	DD CB 05 96	RES 2,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
521	DD CB 05 9E	RES 3,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
522	DD CB 05 A6	RES 4,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
523	DD CB 05 AE	RES 5,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
524	DD CB 05 B6	RES 6,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
525	DD CB 05 BE	RES 7,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
526	DD CB 05 C6	SET 0,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
527	DD CB 05 CE	SET 1,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
528	DD CB 05 D6	SET 2,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
529	DD CB 05 DE	SET 3,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
530	DD CB 05 E6	SET 4,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
531	DD CB 05 EE	SET 5,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
532	DD CB 05 F6	SET 6,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
533	DD CB 05 FE	SET 7,(IX+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
534	DD E1	POP IX	..X.X...	4	14 (4,4,3,3)	5.60	
535	DD E3	EX (SP),IX	..X.X...	6	23 (4,4,3,4,3,5)	9.20	
536	DD E5	PUSH IX	..X.X...	4	15 (4,5,3,3)	6.00	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
537	DD E9	JP (IX)	..X.X...	2	8 (4,4)	3.20	
538	DD F9	LD SP,IX	..X.X...	2	10 (4,6)	4.00	
539	DE 20	SBC A,N	!!X!XV1!	2	7 (4,3)	2.80	
540	DF	RST 18H	..X.X...	3	11 (5,3,3)	4.40	
541	E0	RET PO	..X.X...	3	11 (5,3,3)	4.40	DACA P =0
				1	5 (5)	2.00	DACA P =1
542	E1	POP HL	..X.X...	3	10 (4,3,3)	4.00	
543	E2 BB AA	JP PO,NN	..X.X...	3	10 (4,3,3)	4.00	
544	E3	EX (SP),HL	..X.X...	5	19 (4,3,4,4,3,5)	7.60	
545	E4 BB AA	CALL PO,NN	..X.X...	5	17 (4,3,4,4,3,3)	6.80	DACA P =0
				3	10 (4,3,3)	4.00	DACA P =1
546	E5	PUSH HL	..X.X...	3	11 (5,3,3)	4.40	
547	E6 20	AND N	!!X1XP00	2	7 (4,3)	2.80	
548	E7	RST 20H	..X.X...	3	11 (5,3,3)	4.40	
549	E8	RET PE	..X.X...	3	11 (5,3,3)	4.40	DACA P =1
				1	5 (5)	2.00	DACA P =0
550	E9	JP (HL)	..X.X...	1	4 (4)	1.60	
551	EA BB AA	JP PE,NN	..X.X...	3	10 (4,3,3)	4.00	
552	EB	EX DE,HL	..X.X...	1	4 (4)	1.60	
553	EC BB AA	CALL PE,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA P =1
				3	10 (4,3,3)	4.00	DACA P =0
554	ED 40	IN B,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
555	ED 41	OUT (C),B	..X.X...	3	12 (4,4,4)	4.80	
556	ED 42	SBC HL,BC	!!X!XV1!	4	15 (4,4,4,3)	6.00	
557	ED 43 BB AA	LD (NN),BC	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
558	ED 44	NEG	!!X!XV1!	2	8 (4,4)	3.20	
559	ED 45	RETN	..X.X...	4	14 (4,4,3,3)	5.60	
560	ED 46	IM 0	..X.X...	2	8 (4,4)	3.20	
561	ED 47	LD I,A	..X.X...	2	9 (4,5)	3.60	
562	ED 48	IN C,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
563	ED 49	OUT (C),C	..X.X...	3	12 (4,4,4)	4.80	
564	ED 4A	ADC HL,BC	!!X!XV0!	4	15 (4,4,4,3)	6.00	
565	ED 4B BB AA	LD BC,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
566	ED 4D	RETI	..X.X...	4	14 (4,4,3,3)	5.60	
567	ED 4F	LD R,A	..X.X...	2	9 (4,5)	3.60	
568	ED 50	IN D,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
569	ED 51	OUT (C),D	..X.X...	3	12 (4,4,4)	4.80	
570	ED 52	SBC HL,DE	!!X!XV1!	4	15 (4,4,4,3)	6.00	
571	ED 53 BB AA	LD (NN),DE	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
572	ED 56	IM 1	..X.X...	2	8 (4,4)	3.20	
573	ED 57	LD A,I	!!X0X!0.	2	9 (4,5)	3.60	
574	ED 58	IN E,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
575	ED 59	OUT (C),E	..X.X...	3	12 (4,4,4)	4.80	
576	ED 5A	ADC HL,DE	!!X!XV0!	4	15 (4,4,4,3)	6.00	
577	ED 5B BB AA	LD DE,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
578	ED 5E	IM 2	..X.X...	2	8 (4,4)	3.20	
579	ED 5F	LD A,R	!!X0X!0.	2	9 (4,5)	3.60	
580	ED 60	IN H,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
581	ED 61	OUT (C),H	..X.X...	3	12 (4,4,4)	4.80	
582	ED 62	SBC HL,HL	!!X!XV1!	4	15 (4,4,4,3)	6.00	
583	ED 63 BB AA	LD (NN),HL	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
584	ED 67	RRD	!!X0XP0.	5	18 (4,4,3,4,3)	7.20	
585	ED 68	IN L,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
586	ED 69	OUT (C),L	..X.X...	3	12 (4,4,4)	4.80	
587	ED 6A	ADC HL,HL	!!X!XV0!	4	15 (4,4,4,3)	6.00	
588	ED 6B BB AA	LD HL,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
589	ED 6F	RLD	!!X0XP0.	5	18 (4,4,3,4,3)	7.20	
590	ED 72	SBC HL,SP	!!X!XV1!	4	15 (4,4,4,3)	6.00	
591	ED 73 BB AA	LD (NN),SP	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
592	ED 78	IN A,(C)	!!X!XP0.	3	12 (4,4,4)	4.80	
593	ED 79	OUT (C),A	..X.X...	3	12 (4,4,4)	4.80	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
594	ED 7A	ADC HL,SP	!!X!XVO!	4	15 (4,4,4,3)	6.00	
595	ED 7B BB AA	LD SP,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
596	ED A0	LDI	..X0X!0.	4	16 (4,4,3,5)	6.40	
597	ED A1	CPI	!!X!X!1.	4	16 (4,4,3,5)	6.40	
598	ED A2	INI	X!XXXX1.	4	16 (4,5,4,3)	6.40	
599	ED A3	OUTI	X!XXXX1.	4	16 (4,5,3,4)	6.40	
600	ED A8	LDD	..X0X!0.	4	16 (4,4,3,5)	6.40	
601	ED A9	CPD	!!X!X!1.	4	16 (4,4,3,5)	6.40	
602	ED AA	IND	X!XXXX1.	4	16 (4,5,4,3)	6.40	
603	ED AB	OUTD	X!XXXX1.	4	16 (4,5,3,4)	6.40	
604	ED B0	LDIR	..X0X00.	5	21 (4,4,3,5,5)	8.40	DACA BC # 0
				4	16 (4,4,3,5)	6.40	DACA BC = 0
605	ED B1	CPIR	!!X!X!1.	5	21 (4,4,3,5,5)	8.40	DACA BC#0 SI A#(HL)
				4	16 (4,4,3,5)	6.40	DACA BC=0 SAU A=(HL)
606	ED B2	INIR	X1XXXX1.	5	21 (4,5,4,3,5)	8.40	DACA B # 0
				4	16 (4,5,4,3)	6.40	DACA B = 0
607	ED B3	OTIR	X1XXXX1.	5	21 (4,5,3,4,5)	8.40	DACA B # 0
				4	16 (4,5,3,4)	6.40	DACA B = 0
608	ED B8	LDDR	..X0X00.	5	21 (4,4,3,5,5)	8.40	DACA BC # 0
				4	16 (4,4,3,5)	6.40	DACA BC = 0
609	ED B9	CPDR	!!X!X!1.	5	21 (4,4,3,5,5)	8.40	DACA BC#0 SI A#(HL)
				4	16 (4,4,3,5)	6.40	DACA BC=0 SAU A=(HL)
610	ED BA	INDR	X1XXXX1.	5	21 (4,5,4,3,5)	8.40	DACA B # 0
				4	16 (4,5,4,3)	6.40	DACA B = 0
611	ED BB	OTDR	X1XXXX1.	5	21 (4,5,3,4,5)	8.40	DACA B # 0
				4	16 (4,5,3,4)	6.40	DACA B = 0
612	EE 20	XOR N	!!XOXPOO	2	7 (4,3)	2.80	
613	EF	RST 28H	..X.X...	3	11 (5,3,3)	4.40	
614	F0	RET P	..X.X...	3	11 (5,3,3)	4.40	DACA S =0
				1	5 (5)	2.00	DACA S =1
615	F1	POP AF	!!!!!!!!	3	10 (4,3,3)	4.00	
616	F2 BB AA	JP P,NN	..X.X...	3	10 (4,3,3)	4.00	
617	F3	DI	..X.X...	1	4 (4)	1.60	
618	F4 BB AA	CALL P,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA S =0
				3	10 (4,3,3)	4.00	DACA S =1
619	F5	PUSH AF	..X.X...	3	11 (5,3,3)	4.40	
620	F6 20	OR N	!!XOXPOO	2	7 (4,3)	2.80	
621	F7	RST 30H	..X.X...	3	11 (5,3,3)	4.40	
622	F8	RET M	..X.X...	3	11 (5,3,3)	4.40	DACA S =1
				5	5 (5)	2.00	DACA S =0
623	F9	LD SP,HL	..X.X...	1	6 (6)	2.40	
624	FA BB AA	JP M,NN	..X.X...	3	10 (4,3,3)	4.00	
625	FB	EI	..X.X...	1	4 (4)	1.60	
626	FC BB AA	CALL M,NN	..X.X...	5	17 (4,3,4,3,3)	6.80	DACA S =1
				3	10 (4,3,3)	4.00	DACA S =0
627	FD 09	ADD IY,BC	..X!X.0!	4	15 (4,4,4,3)	6.00	
628	FD 19	ADD IY,DE	..X!X.0!	4	15 (4,4,4,3)	6.00	
629	FD 21 BB AA	LD IY,NN	..X.X...	4	14 (4,4,3,3)	5.60	
630	FD 22 BB AA	LD (NN),IY	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
631	FD 23	INC IY	..X.X...	2	10 (4,6)	4.00	
632	FD 29	ADD IY,IY	..X!X.0!	4	15 (4,4,4,3)	6.00	
633	FD 2A BB AA	LD IY,(NN)	..X.X...	6	20 (4,4,3,3,3,3)	8.00	
634	FD 2B	DEC IY	..X.X...	2	10 (4,6)	4.00	
635	FD 34 05	INC (IY+IND)	!!X!XV0.	6	23 (4,4,3,5,4,3)	9.20	
636	FD 35 05	DEC (IY+IND)	!!X!XV1.	6	23 (4,4,3,5,4,3)	9.20	
637	FD 36 05 20	LD (IY+IND),N	..X.X...	5	19 (4,4,3,5,3)	7.60	
638	FD 39	ADD IY,SP	..X!X.0!	4	15 (4,4,4,3)	6.00	
639	FD 46 05	LD B,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
640	FD 4E 05	LD C,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
641	FD 56 05	LD D,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
642	FD 5E 05	LD E,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	

nr. crt	cod	mnemonica	flag sz h pnc	nr. cic	nr. tacti	timp 2.5m	comentarii
643	FD 66 05	LD H,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
644	FD 6E 05	LD L,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
645	FD 70 05	LD (IY+IND),B	..X.X...	5	19 (4,4,3,5,3)	7.60	
646	FD 71 05	LD (IY+IND),C	..X.X...	5	19 (4,4,3,5,3)	7.60	
647	FD 72 05	LD (IY+IND),D	..X.X...	5	19 (4,4,3,5,3)	7.60	
648	FD 73 05	LD (IY+IND),E	..X.X...	5	19 (4,4,3,5,3)	7.60	
649	FD 74 05	LD (IY+IND),H	..X.X...	5	19 (4,4,3,5,3)	7.60	
650	FD 75 05	LD (IY+IND),L	..X.X...	5	19 (4,4,3,5,3)	7.60	
651	FD 77 05	LD (IY+IND),A	..X.X...	5	19 (4,4,3,5,3)	7.60	
652	FD 7E 05	LD A,(IY+IND)	..X.X...	5	19 (4,4,3,5,3)	7.60	
653	FD 86 05	ADD A,(IY+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
654	FD 8E 05	ADC A,(IY+IND)	!!X!XVO!	5	19 (4,4,3,5,3)	7.60	
655	FD 96 05	SUB (IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
656	FD 9E 05	SBC A,(IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
657	FD A6 05	AND (IY+IND)	!!X1XP00	5	19 (4,4,3,5,3)	7.60	
658	FD AE 05	XOR (IY+IND)	!!XOXP00	5	19 (4,4,3,5,3)	7.60	
659	FD B6 05	OR (IY+IND)	!!XOXP00	5	19 (4,4,3,5,3)	7.60	
660	FD BE 05	CP (IY+IND)	!!X!XV1!	5	19 (4,4,3,5,3)	7.60	
661	FD CB 05 06	RLC (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
662	FD CB 05 0E	RRC (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
663	FD CB 05 16	RL (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
664	FD CB 05 1E	RR (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
665	FD CB 05 26	SLA (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
666	FD CB 05 2E	SRA (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
667	FD CB 05 3E	SRL (IY+IND)	!!XOXP0!	6	23 (4,4,3,5,4,3)	9.20	
668	FD CB 05 46	BIT 0,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
669	FD CB 05 4E	BIT 1,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
670	FD CB 05 56	BIT 2,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
671	FD CB 05 5E	BIT 3,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
672	FD CB 05 66	BIT 4,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
673	FD CB 05 6E	BIT 5,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
674	FD CB 05 76	BIT 6,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
675	FD CB 05 7E	BIT 7,(IY+IND)	X!X1XX0.	5	20 (4,4,3,5,4)	8.00	
676	FD CB 05 86	RES 0,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
677	FD CB 05 8E	RES 1,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
678	FD CB 05 96	RES 2,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
679	FD CB 05 9E	RES 3,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
680	FD CB 05 A6	RES 4,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
681	FD CB 05 AE	RES 5,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
682	FD CB 05 B6	RES 6,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
683	FD CB 05 BE	RES 7,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
684	FD CB 05 C6	SET 0,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
685	FD CB 05 CE	SET 1,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
686	FD CB 05 D6	SET 2,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
687	FD CB 05 DE	SET 3,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
688	FD CB 05 E6	SET 4,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
689	FD CB 05 EE	SET 5,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
690	FD CB 05 F6	SET 6,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
691	FD CB 05 FE	SET 7,(IY+IND)	..X.X...	6	23 (4,4,3,5,4,3)	9.20	
692	FD E1	POP IY	..X.X...	4	14 (4,4,3,3)	5.60	
693	FD E3	EX (SP),IY	..X.X...	6	23 (4,4,3,4,3,5)	9.20	
694	FD E5	PUSH IY	..X.X...	4	15 (4,5,3,3)	6.00	
695	FD E9	JP (IY)	..X.X...	2	8 (4,4)	3.20	
696	FD F9	LD SP,IY	..X.X...	2	10 (4,6)	4.00	
697	FE 20	CP N	!!X!XV1!	2	7 (4,3)	2.80	
698	FF	RST 38H	..X.X...	3	11 (5,3,3)	4.40	

G

INSTRUCȚIUNI ASCUNSE ALE MICROPROCESORULUI Z80

Studiind setul de coduri ale instrucțiunilor microprocesorului **Z80** pot fi observate goluri. Inexistența unor instrucțiuni specificate i-a pus pe gînduri pe unii utilizatori ai acestei capsule, care abordînd problema empiric au descoperit o serie întreagă de coduri utilizabile, nedenumite și nepublicate de către fabricanți. Prima descoperire cunoscută nouă a realizat-o cercet. șt. Szász Detre la ITCI Filiala Cluj-Napoca. Codurile identificate și testate de dînsul se cifrează la 96. Publicate intern, ele au putut fi folosite de către programatorii institutului.

Se pune întrebarea de ce fabricanții, sau măcar proiectanții microprocesorului **Z80,** nu au publicat existența acestor instrucțiuni ? Întrebarea rămîne deocamdată deschisă. Încercările făcute în diverse țări pe loturi foarte variate (provenind de la diverse firme) au demonstrat că aceste instrucțiuni nu se execută aleator. Rezultă că ele își au originea în însuși proiectul inițial (măștile de fabricație) al circuitului integrat **Z80.**

Fenomenul a fost între timp descoperit și la alte microprocesoare (ex.**I8086**) și s-a cristalizat o atitudine destul de unitară față de fenomenul instrucțiunilor ascunse.

O povestim pe scurt :

1. Existența unor instrucțiuni ascunse la unele procesoare este certă.

2. Cert este că fabricanții nu testează aceste instrucțiuni și deci nici nu-și asumă nici o răspundere pentru buna funcționare a microprocesorului în cazul folosirii lor.

3. De obicei nu există asambloare, dezasambloare care să recunoască aceste instrucțiuni, care fiind descoperite în diverse părți ale lumii poartă cele mai diverse mnemonici pentru unul și același cod. Utilizatorul poate folosi aceste instrucțiuni sub rezerva că ar putea apare un procesor al unui nou fabricant, care să nu suporte aceste instrucțiuni, caz în care software-ul va trebui să fie reconsiderat. (Acest din urmă eveniment este puțin probabil în cazul microprocesorului **Z80,** el aflîndu-se în fabricație deja de mai mult de un deceniu).

G.1. Prezentarea fenomenului. Mnemonici propuse.

Instructiunile publicate de Szász Detre şi totodată de către autor se referă la două fenomene :

1. — posibilitatea tratării celor doi regiştri index **IX, IY** asemănător cu regiştri de uz comun, cum ar fi **HL**. S-a demonstrat că cei doi octeţi ai unui registru index pot fi trataţi individual, efectuîndu-se transferuri şi operaţii aritmetice/logice cu ele.

Propunem ca nume de argumente : **XH** — octetul superior al registrului IX
XL — octetul inferior al registrului IY
YH — octetul superior al registrului IY
YL — octetul inferior al registrului IY

Aceşti regiştri vor putea efectua următoarele 10 tipuri de operaţii aritmetice/logice cu **A**:

ADC,ADD,AND,CP,DEC,INC,OR,SBC,SUB,XOR

obţinîndu-se în total 40 de instrucţiuni. Codurile acestor instrucţiuni se obţin folosind unul din cele 2 coduri rezervate (**DD** pentru **X** şi **FD** pentru **Y**) şi codul instrucţiunilor similare care folosesc regiştri **H** sau **L**.

Ex. :

ADC	A,H	are codul 8C
ADC	A,XH	are codul DD 8C
ADC	A,YH	are codul FD 8C
XOR	L	are codul AD
XOR	XL	are codul DD AD
XOR	YL	are codul FD AD

Setul se completează cu 40 de instrucţiuni de transfer interregistru.

Ex. :

LD	E,XH	(DD 5C)
LD	YL,A	(FD 6F)

În fine **XH,XL, YH şi YL** pot fi încărcaţi cu valori prin instrucţiuni imediate cum ar fi :

LD	XH,nn	(DD 2C nn)

2. — posibilitatea efectuării unui nou tip de instrucţiune de deplasare la stînga : conţinutul registrului sau al celulei de memorie selectate se deplasează cu o poziţie la stînga (**D7 în Cy**), iar pe bitul cel mai puţin semnificativ se inserează 1. Această instrucţiune execută de fapt operaţia (2 * R+1), care poate fi utilă mai ales la implementarea unor funcţii matematice.

Menmonica propusă este **SLH** (Shift Left High)

Ex. :

SLH	B	are codul CB 30
SLH	(IX+IND)	are codul DD CB IND 36

Totalul acestor instrucțiuni se cifrează la 10.

Obținem astfel un set de 94 instrucțiuni ale căror date de funcționare se regăsesc în tabela G.2 (pag. 229—230).

În afara acestor instrucțiuni, testate de autori, s-a publicat o nouă clasă numărînd 364 de instrucțiuni.

Aceste instrucțiuni efectuează operații duble :

— operație **ROT/SHIFT** urmată de transfer de date

— operație **SET/RES** urmată de transfer de date

Ele au fost descoperite pe aceeași cale intuitivă pornind de la observația că în lista originală a instrucțiunilor **Z80** secvența de cod **DD CB IND x**, cuprinde doar posibilitatea utilizării unor valori x care au pe digitul cel mai puțin semnificativ cifra **6** sau **E**.

Ex. :

DD CB IND 06 înseamnă **RLC (IX+IND)**

Încercînd diverse valori pentru **x**, altele decît cele prevăzute, s-a descoperit că ele generează activități interesante : rotirea celulei de memorie este urmată de transferul datei rotite într-un registru selectat. S-a demonstrat astfel că setul publicat de fabricanți este un caz particular, în care transferul se efectuează în însăși celula adresată prin **(IX+IND)** sau **(IY+IND)**.

Rezultatul instrucțiunii poate fi dirijat în afara celulei de memorie indexate în unul din regiștri **A,B,C,D,E,H** sau **L**.

Pe această cale s-au identificat funcții similare pentru instrucțiuni de tip **RES** și **SET**.

Ex. :

RR (IY+IND),E	are codul **FD CB IND 1B**
SET 5, (IX+IND),A	are codul **DD CB IND EF**

În tabela G.1. sintetizăm aceste instrucțiuni ascunse, menționînd că primele 8 coduri din coloana **(IX+d)** sînt recunoscute și publicate de firme.

Formatul general al codurilor va fi :

pentru **(IX+d)** : **DD CB d x**

pentru **(IY+d)** : **FD CB d x**

— valoarea lui **x** se ia din tabela G.1.

— **d** reprezintă deplasamentul (indicele).

Remarcăm absența codurilor **40 — 7F** care ar reprezenta instrucțiuni de tip **BIT**. Lăsăm pe seama cititorului să încerce acest set de coduri.

Autorul a testat cîteva din instrucțiunile tabelei C1. Ele par a fi într-adevăr corecte.

Astfel totalul instrucțiunilor acceptate de microprocesorul **Z80** crește cu : 94 + 364 = **458**, față de cele **696** publicate.

Tabela G.1. **Instrucțiuni de manipulare a celulelor de memorie indexate, urmate de transfer în registru**

	r	B	C	D	E	H	L	(IX+IND)	A
RLC	(IX+d),r	00	01	02	03	04	05	06	07
RRC	(IX+d),r	08	09	0A	0B	0C	0D	0E	0F
RL	(IX+d),r	10	11	12	13	14	15	16	17
RR	(IX+d),r	18	19	1A	1B	1C	1D	1E	1F
SLA	(IX+d),r	20	21	22	23	24	25	26	27
SRA	(IX+d),r	28	29	2A	2B	2C	2D	2E	2F
SLH	(IX+d),r	30	31	32	33	34	35	36	37
SRL	(IX+d),r	38	39	3A	3B	3C	3D	3E	3F
RES	0,(IX+d),r	80	81	82	83	84	85	86	87
RES	1,(IX+d),r	88	89	8A	8B	8C	8D	8E	8F
RES	2,(IX+d),r	90	91	92	93	94	95	96	97
RES	3,(IX+d),r	98	99	9A	9B	9C	9D	9E	9F
RES	4,(IX+d),r	A0	A1	A2	A3	A4	A5	A6	A7
RES	5,(IX+d),r	A8	A9	AA	AB	AC	AD	AE	AF
RES	6,(IX+d),r	B0	B1	B2	B3	B4	B5	B6	B7
RES	7,(IX+d),r	B8	B9	BA	BB	BC	BD	BE	BF
SET	0,(IX+d),r	C0	C1	C2	C3	C4	C5	C6	C7
SET	1,(IX+d),r	C8	C9	CA	CB	CC	CD	CE	CF
SET	2,(IX+d),r	D0	D1	D2	D3	D4	D5	D6	D7
SET	3,(IX+d),r	D8	D9	DA	DB	DC	DD	DE	DF
SET	4,(IX+d),r	E0	E1	E2	E3	E4	E5	E6	E7
SET	5,(IX+d),r	E8	E9	EA	EB	EC	ED	EE	EF
SET	6,(IX+d),r	F0	F1	F2	F3	F4	F5	F6	F7
SET	7,(IX+d),r	F8	F9	FA	FB	FC	FD	FE	FF

ÎNTREPRINDEREA POLIGRAFICĂ CLUJ,
Municipiul Cluj-Napoca, B-dul Lenin, nr. 146.
cd. nr. 425/1988

1— 2. L. Dumitrașcu — **Învățăm microelectronică interactivă conversînd în BASIC. Totul despre... BASIC în 14 conversații și 7 sinteze pe Felix C, CORAL, INDEPENDENT, Felix PC, M 118, TPD, HC. 85, aMIC, PRAE, COMMODORE, AMSTRAD și compatibile,** vol. 1 și 2.

3— 4. A. Petrescu și colectiv IPB, ITCI, Fabrica de calculatoare, Liceul Dimitrie Cantemir, CNOP — **ABC de calcul electronic. Totul despre... HC 85,** vol. 1, și vol. 2, **(o parte a tirajului cu 2 casete cu programe, acționînd calculatoare personale HC 85 și compatibile SINCLAIR SPECTRUM)**

5— 6. A. Tănăsescu și colectiv IPB, ISPIF — **Grafică asistată de calculator. Programe Fortran pe minicalculatoare, pentru reprezentări geometrice,** vol. 1 și vol. 2.

7—12. Colective largi — **Automatică, management, calculatoare (AMC). Serie continuă de instruire, informare, sinteze, cercetări aplicative în sisteme electronice, automate, informatice, de conducere,** volumele 56—51. **Echipamente electronice și tehnică de calcul — manuale de utilizare. Calculatoare personale, programe ș.a.**

13. A. Davidoviciu și colectiv ITCI, ASE — **Sistemul de operare MIX și limbajul MACRO pentru minicalculatoare CORAL/INDEPENDENT,** 2 volume.

14—15. I. Văduva, V. Baltac, Florescu V și colectiv ASE, ITCI — **Informatică economică,** 2 volume

16. Rusu O., Brudaru I. — **Echilibrarea liniilor flexibile.**

17. P. Constantinescu — **Sinergia, informația și geneza sistemelor.**

Se difuzează prin unitățile centrelor de librării, spre care se îndrumă întreprinderile și cititorii.

PENTRU ACESTE CARȚI SE POT FACE, TOTUȘI, ȘI COMENZI FERME LA EDITURA TEHNICĂ, PIAȚA SCÎNTEII 1, BUCUREȘTI.

Comenzile întreprinderilor se semnează de director și contabil șef, cele ale cititorilor individuali au indicată adresa exactă. Comenzile se trimit de editură la centrele de librării, cu indicarea unor priorități de satisfacere a lor. Plata nu se face decît la primirea exemplarelor.

I-Vol. I-II.

• Aveți in față o carte intr-adevăr mult așteptată: prima carte editată în România însoțită (o parte a tirajului) de o casetă magnetică ce acționează calculatoare personale.

• Este adresată celor care studiază sau lucrează în electronică – automatizări – informatică – tehnică de calcul, cum și tuturor utilizatorilor actuali sau potențiali de microprocesoare, din toate ramurile economiei naționale.

• Cunoașterea funcționării intime a microprocesorului Z 80 (cel mai răspindit în țară drept „creier electronic" al echipamentelor de calcul și de automatizare, alături de mai vechiul I 8080, pe care il depășește cu mult, dar cu care este compatibil unilateral), ca și proiectarea electronică a utilajelor dotate cu acest microprocesor este posibilă atît pentru cititorii care au acces la un calculator personal PRAE sau aMIC, cît și pentru cei care nu au. Aceasta, pentru că peste 200 din secvențele esențiale pe care simulatorul VISIBLE–Z 80, aflat pe caseta din casetofon, le redă dinamic pe ecranul monitorului sau televizorului atașat calculatorului personal sînt selectate și imprimate în carte, cu numeroase explicații și comentarii.

ISBN 973-31-0008-0
ISBN 973-31-0007-2